2002

Logserver monitor for managing log messages of applications

Lilin Zhu

LOGSERVER MONITOR FOR MANAGING LOG MESSAGES OF APPLICATIONS

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Lilin Zhu

June 2002

LOGSERVER MONITOR FOR MANAGING LOG MESSAGES OF APPLICATIONS

A Project

Presented to the

Faculty of

California State University,

San Bernardino

by

Lilin Zhu

June 2002

Approved by:

[redacted]

Dr. Richard Botting, Chair, Computer Science

May 15, 2002

Date

[redacted]

Dr. Ernesto Gomez

[redacted]

Dr. David Turner

ABSTRACT

Logging represents an important component of a software development cycle. Almost every large software application includes its own logging or tracing API. This project creates a graphical user interface (GUI) to manage and display log information in a distributed environment.

This project uses Java "Swing" components to create the graphic user interface and uses Log4j APIs, a logging package for Java, to log messages from a distributed environment. The main user interface is composed of a menu bar and a tabbed window with three tabs: "File View," "Logging View," and "Email." The log messages can be displayed in the "Logging View" tab dynamically or can be saved in a log file.

The "File View" tab has a table and list area. The name and creation time of the existing log files can be displayed in the list area. The contents of the selected file can be displayed in the table. The table can be sorted by column and the user can turn on or off any column(s). The user can choose to display messages by types such as fatal, error, or info. The messages in the table can be color-coded by message type. The user can also change font

and search the messages in the table. There are also printing, copy, and paste functions on the user interface.

The "Logging View" tab has a text area and list area. The list area lists the names of all clients that are currently connected or have been connected with the Log Server. The text area displays the log message received from all the clients. When the user selects a client from the list, the message from that client will be displayed in the new window.

The "Email" tab is designed for the user to send email to clients to inform them if errors occur.

ACKNOWLEDGMENTS

I would like to thank Dr. Ernesto Gomez and Dr. David Turner for serving on my committee and their insightful comments on the draft of this report. I am particularly grateful to Dr. Richard Botting, my project advisor, for his patience, advice, guidance, detailed suggestions and corrections of the design, implementation, and report of this project.

I would also like to thank Mr. Jay Theodore of ESRI for getting me started on this project.

Last but not least, I would like to thank to my husband for his constant support for my continuing education in general and for this project in particular. He also spent many hours playing games with my daughter so I could work on this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

Purpose of the Project

Logging is an important component of a software development cycle as well as for diagnostics of performance and monitoring of the software after deployment. It can provide precise context about a run of the application. This project is a graphical user interface for managing log information. The LogServer Monitor can help you easily view information about how an application is running and what kind of problem the application is having.

The idea of writing a Java LogServer Monitor started when the author collaborated with ESRI's ArcIMS Middleware team. ESRI (Environmental Systems Research Inc.) is a geographic information system (GIS) software company headquartered in Redlands, CA. ESRI wanted an application to display and manage the log messages created from its server programs in its Internet map server software, ArcIMS. This project started with that purpose in mind. Therefore, this is an attempt to solve a real-world software problem. ESRI's ArcIMS middleware is primarily developed in Java. Thus, Java is a logical choice for the application.

Scope of the Project

The LogServer Monitor provides a graphical user interface for the display and management of logged information from a distributed environment such as ESRI's ArcIMS Server components. The LogServer Monitor uses Java Swing components to create user interfaces and uses the Log4J APIs, a Java logging package to log the information. Log4j is an open source project, which allows developers to control which log statements are output with arbitrary granularity. It's fully configurable at runtime by using external configuration files. The logging behavior can be controlled by editing a configuration file without touching the application code [5].

In the application, the log messages can be either displayed dynamically in real time or written in the log files and then displayed in a table. This project has a LogServer Monitor window. From this window, a log request can be selected. The user can choose to view from the log file or view the log message dynamically.

The main user interface is composed of a menu bar and a tabbed window with three tabs: "File View," "Logging View," and "Email." The "File View" tab has a table and a list area. The name and creation time of the existing log files can be displayed in the list. There is an "Update

File" button on this tab. The user can update the file list by clicking on this button. The user can select which file to be displayed in the table from a list of available log files. The user can also open the log files from the file menu on the menu bar if he/she uses Microsoft Windows platform.

The "Logging View" tab has a list and a text area. The list displays the names of all clients that are currently or have been connected with log server. There is an "Update Clients" button on this tab. The user can update the client list by clicking on this button. All the log messages from all the clients are displayed in the text area dynamically. When the user selects a client from the list, that client's messages should be displayed in a new window dynamically.

The Email tab is designed for the user to send email through the network to inform the clients if errors occur.

Significance of the Project

Logging is important in software development because it enables developers to quickly see when a problem has occurred in the code without having to step through the code line-by-line. It is also crucial for the quality assurance personnel to create clear and specific

bug reports so developers can immediately zero in on the problem.

In addition, logging is crucial when an application is deployed. Once the application is deployed to a client site you can't start your debugger and you can't edit the code to determine what has gone wrong. Logging facilitates software servicing and maintenance at a client site by producing log reports for analysis by end users, system administrators, field service engineers, and software development teams.

However, a typical log file is a flat file with a list of hundreds of thousands or millions of lines of text. It is almost impossible for a user to wade through the text and find a particular piece of information. The user needs a set of tools to view what is stored in the log file. LogServer monitor is created precisely for this purpose. It provides a set of tools to allow the user to display log information in a variety of ways. So the user can easily use what is stored in the log files. In other words, the LogServer Monitor makes the inaccessible flat log file accessible and useful.

Limitation of the Project

There is a table in the "File View" Tab for displaying messages from log files. The table has a fixed number of columns for certain parts of a log message. This requires that the log files have certain format. The log file format should be defined as: %d %-5p [%t] %c - %m%n (refer to table 1. %: the beginning of the format. d: the date of the logging event. p: the priority of the logging event. t: the name of the thread that generates the logging event. c: the category of the logging event. m: logging message. n: line separator. 5: means the priority of the logging event should be left justified to a width of five characters.). This is a limitation of this project.

Another limitation of this project is that the email function requires that the local host be running an SMPT server.

CHAPTER TWO

REVIEW OF RELATED WORK

Related Logging Software

Silicon Graphics' System Log File Viewer

Sysmon is a system log file viewer developed by Silicon Graphics Inc. (SGI) as a part of System Monitor, an error reporting system for the IRIX Interactive Desktop. The sysmon utility allows a user to browse the system log file generated by IRIX.

The sysmon utility simplifies the original eight system priorities into four priority levels: critical, error, warning, and info. The following diagram shows a sysmon System Log Viewer window:

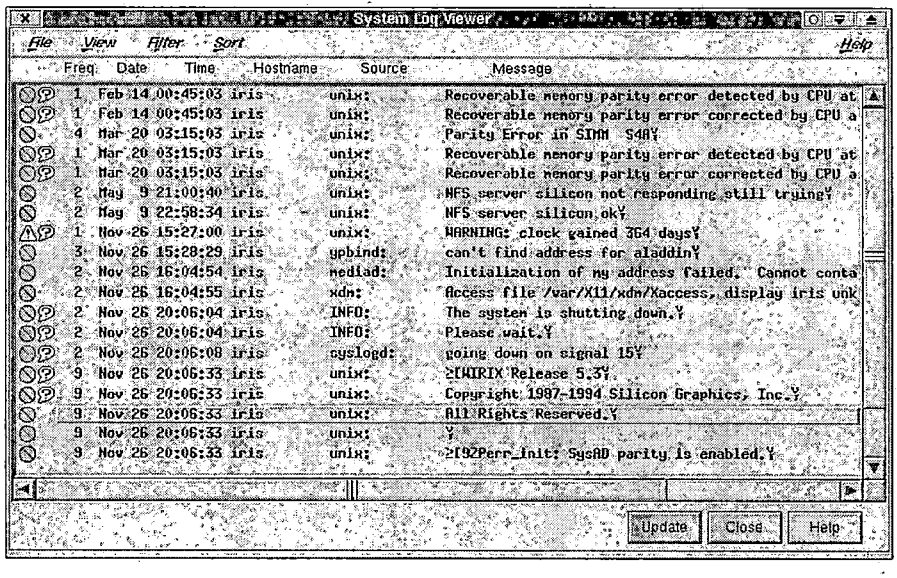


Figure 1. Sysmon Log Viewer Window

The user can choose View, Filter and Sort options through the pull down menus on this window.

Some functions in the LogServer Monitor user interface were influenced by the sysmon utility. For example, The LogServer Monitor has a similar window as sysmon does.

Sysmon is a logging package for a specific operating system. By contrast, LogServer Monitor can be used to view and manage log information for any software application as long as it uses the Log file format. LogServer Monitor also uses XML technology but sysmon does not.

Another Graphic Log Viewer

Chainsaw is a graphic user interface for viewing log files. It also uses Log4J API. The log messages are dynamically displayed in a table. The interface allows a user to filter messages by their priorities. Chainsaw has influenced the functional design of the LogServer Monitor. For example, the LogServer Monitor borrowed the concept of filtering messages by their priorities from Chainsaw. However, Chainsaw has a very simple user interface with limited functions. LogServer Monitor has a more comprehensive user interface and many more functions for the user to explore the log information.

Competing Logging Application Programming Interface

Currently, there are two main competing Logging APIs for Java: Log4j, distributed under the Apache Software License, and Java Logging API by Sun Microsystems. The APIs of Log4j and Java Logging are similar in some respect. For example, they are both based on a named hierarchy. But there are also many differences. This project uses Log4j API because of the perceived advantages of Log4j over Java Logging API at the outset of this project. The following paragraphs list these advantages.

The Log4j is compatible with JDK1.1 and later. Java Logging API requires JDK 1.4.

In Log4j, it is very easy to change the priority of a category. By contrast, it is time consuming to change the priority of a category in Java Logging API.

In Log4j, appenders (output destination) and resource bundles are inherited from the hierarchy. In Java Logging API, a logger logs global handlers (appenders in Log4j). It does not inherit any handlers from the hierarchy.

Log4j has a set of priorities: FATAL, ERROR, WARN, INFO and DEBUG, which are clear and easy to understand. Java Logging API has a set of debugging levels: ALL,

SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST and off, which can be confusing.

Log4j has been around for several years and is being widely used in real world projects. In addition, Log4j has been ported to C++, Python and even C#. Therefore, Log4J is more mature and portable than Java Logging API.

CHAPTER THREE

METHODOLOGY

Software Development Environment

Development Environment

LogServer Monitor was developed using the following software:

1. Microsoft Windows operating system.
2. Sun Microsystems Java Development Kit(JDK) 1.3
3. Log4j (version 1.1.3)
4. Borland JBuilder4

Borland JBuilder4 is a GUI based Java development system. It is easy to compile and run Java programs in Jbuilder4. It has intuitive tools for creating new classes and setting up layouts. JBuilder4 can be used on Solaris, Linux, and Windows 98, NT, and 2000. JBuilder4 is hosted on JDK version 1.3 in order to take advantage of its debugging capabilities and enhanced client-side performance. However, one can still build applications using any previous versions of the JDK in Jbuilder4.

Jbuilder4 is one of the most comprehensive visual development environments for building applications,

applets, JSP/Servlets, JavaBeans, Enterprise JavaBeans and distributed J2EE applications for the Java 2 Platform.

Running Platforms

LogServer Monitor software can be run on the following platforms:

1. Windows operating system
2. Unix/Linux operating system

Language Descriptions

Introduction

The LogServer Monitor uses the Java API, especially the Java "Swing" API, to create the graphic user interface (GUI). It uses Log4j, a Java logging API, to send log messages from client programs to the LogServer Monitor. It uses XSLT style sheets to transform XML files to HTML on the fly when they are loaded in a Web Browser such as Internet Explorer 6.

Java

Java is one of the most popular software development languages. Java is a fully object-oriented language developed by Sun Microsystems. Java programs are capable of running on most popular computer platforms without the need for recompilation. Because of its portability,

multithreading, and networking capabilities, Java is being used for developing "middleware" to communicate between clients and databases and other server resources. Java designers eliminated manual memory allocation and deallocation. They introduced true arrays and eliminated pointer arithmetic. They eliminated multiple inheritances, replacing it with a new notion of interface. All these features in Java eliminated the possibility of creating code with the most common kind of bugs [2]. Sun Microsystems provides an implementation of Java 2 Platform called the Java 2 Software Development Kit(J2SDK), that includes the minimum set of tools you need to write software in Java [1].

Java Logging Package

Log4j, a popular logging package for Java, is an open source project. It allows the developer to control which log statements are output with arbitrary granularity. It is fully configurable at runtime using external configuration files [5]. Log4j has three main components:

- Categories: named entities
- Appenders: output destination
- Layouts: output format

These three types of components work together to log messages according to message type and priority, and to control how these messages are formatted and where they are reported. With Log4j, logging behavior can be controlled by editing a log configuration file, without touching the application binary [5]. Logged messages can be sent to different and multiple output destinations in a user-chosen format. The Log4j environment is configurable programmatically. It also supports configuration through files as well as XML documents [5].

Log4j makes it easy to name categories by software component. This can be accomplished by statically instantiating a category in each class, with the category name equal to the fully qualified name of the class [5].

In Log4j terminology, an appender means an output destination. Currently, appenders exist for the console, files, remote socket servers, etc. [5].

The layout is responsible for formatting the logging request according to the user's wishes. The PatternLayout, part of the Log4j distribution, lets the user specify the output format according to conversion patterns similar to the C programming language format strings. Each conversion

specifier starts with a percent sign (%) and is followed by optional format modifiers and a conversion character.

The conversion character specifies the type of data, e.g. category, priority, data, and thread name. The format modifiers control such things as field width, padding, left and right justification [5]. See table 1 for detail.

Table 1. Log4j Conversion Character Description

Conversion Character	Effect
%	Indicates that the next character or characters will specify that a conversion must take place before output.
c	Used to output the category of the logging event.
C	Used to output the fully qualified class name of the caller issuing the logging request.
d	Used to output the date of the logging event.

F	Used to output the file name where the logging request was issued.
I	Used to output location information of the caller which generated the logging event.
L	Used to output the line number from where the logging request was issued.
m	Used to output the application supplied message associated with the logging event.
M	Used to output the Method name where the logging request was issued.
n	Used to output the platform dependent line separator character or characters.
p	Used to output the priority of the logging event.
r	Used to output the number of milliseconds elapsed since the

	start of the application until the creation of the logging event.
t	Used to output the name of the thread that generated the logging event.
x	Used to output the nested diagnostic context associated with the thread that generated the logging event.

The table is adapted from the Log4j Web site for Log4j technology:

http://jakarta.apache.org/log4j

Extensible Markup Language

XML, the Extensible Markup Language, is a meta-markup language for text documents. It has become popular because its structural markup allows documents to describe their own format and contents. It defines a generic syntax used to mark up data with simple, human readable tags. It provides a standard format for domains as diverse as web sites, electronic data interchange, vector graphics,

genealogy, real estate listings, object serialization, remote procedure calls, and voice mail systems [8].

Extensible Style Sheet Language Transformation

XSLT, Extensible Style sheet Language for Transformations, is a language which can transform XML files into a number of formats, HTML being one of them [4][8]. XSLT is based on finding parts of an XML document that match a series of predefined templates, and then applying transformation and formatting rules to each matched part.

Project Structure

The LogServer Monitor is composed of three parts: "File view," "Logging view," and "Email." The LogServer Monitor receives log information from clients through the socket and displays it either in "File View" or "Logging View." The "Email" provides a tool for the user to communicate with clients (Figure 2).

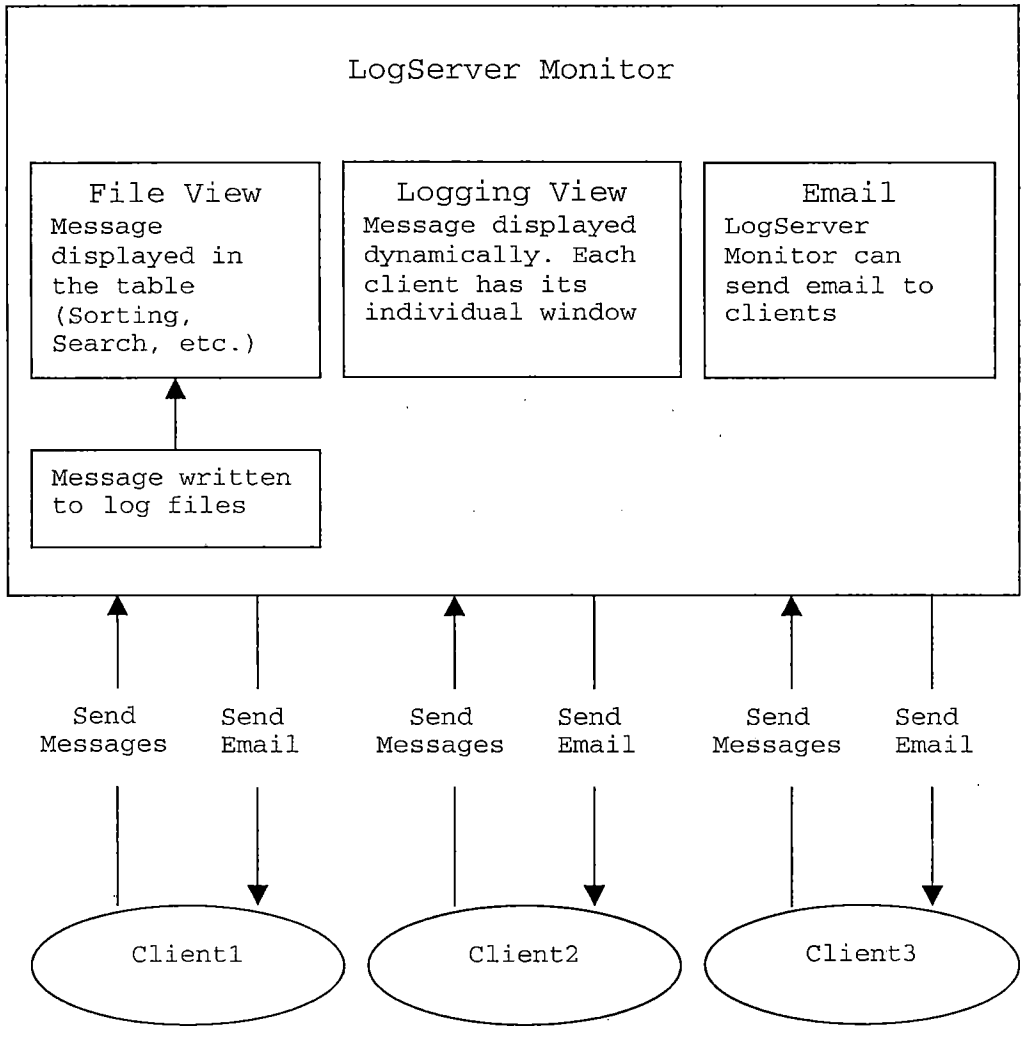


Figure 2. Structure of LogServer Monitor

User Interface Description

Main Window

The LogServer Monitor's main window consists of a menu bar and three tabs. The menu bar has six menus: "File", "Edit", "Format", "Log Level", "Search", and "Columns." The

first tab is the "File View" tab. It has a list area, a button, a table, and a text area. It displays the log messages from the log files (Figure 3).

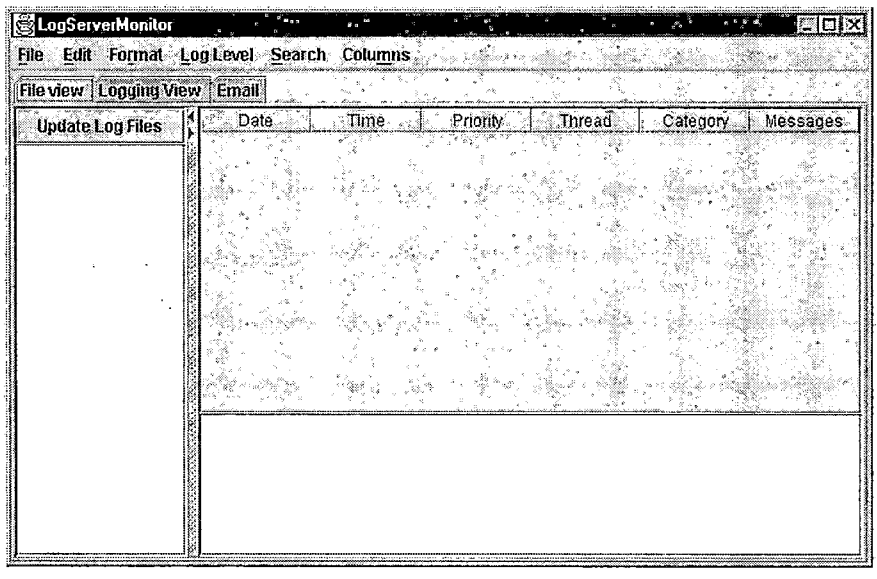


Figure 3. File View Tab

The second tab is the "Logging View" tab. It has a button, a list area, and a text area. It displays the log information dynamically (Figure 4).

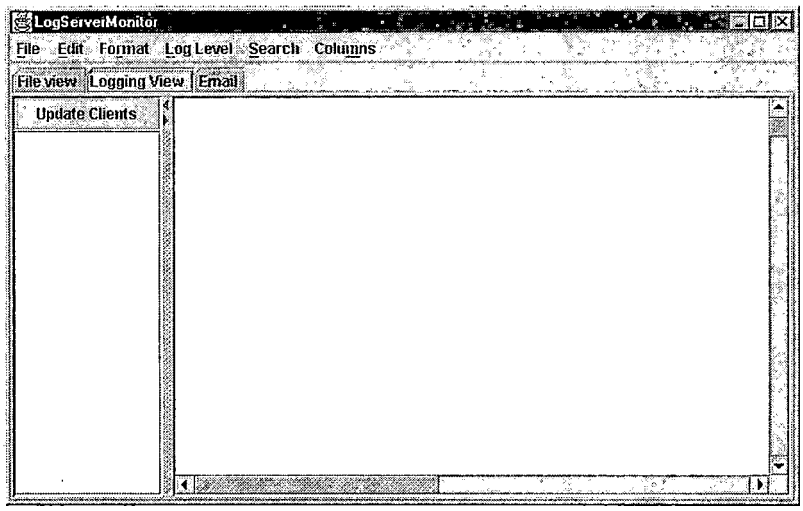


Figure 4. Logging View Tab

The third tab is the "Email" tab. It has three text fields for the user to enter information about the sender ("From"), the receiver ("To"), and the message title ("Subject"). It has a text area for the user to type in the content of the email message (Figure 5).

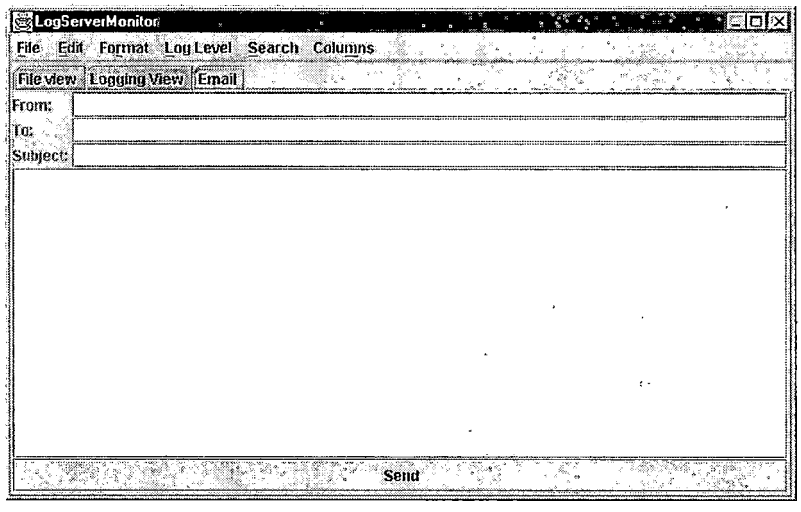


Figure 5. Email Tab

The "File View" Tab

Open Log Files. In the "File View" tab, the user can open log files from the "Open" menu item on the "File" menu. When the user clicks on the "Open" menu item, the open file dialog will pop up. The user can navigate to a directory to open log files. This function can be used only on the Windows platform (Figures 6, 7, and 8).

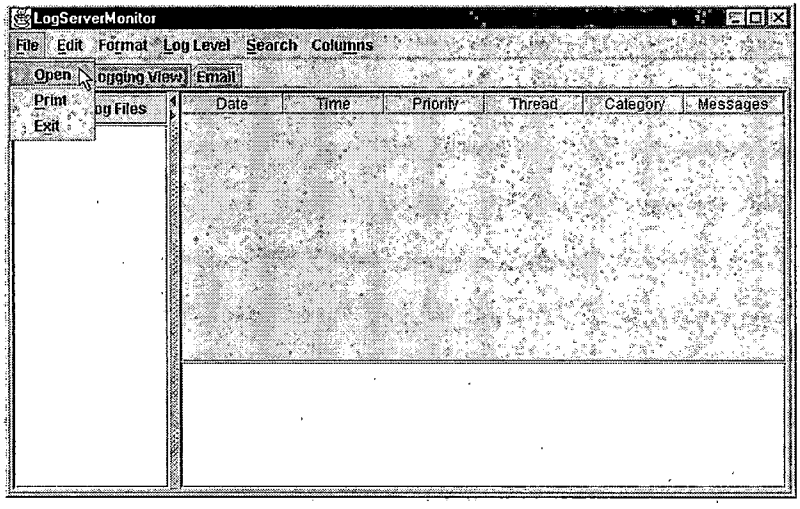


Figure 6. Open a File from the File Menu.

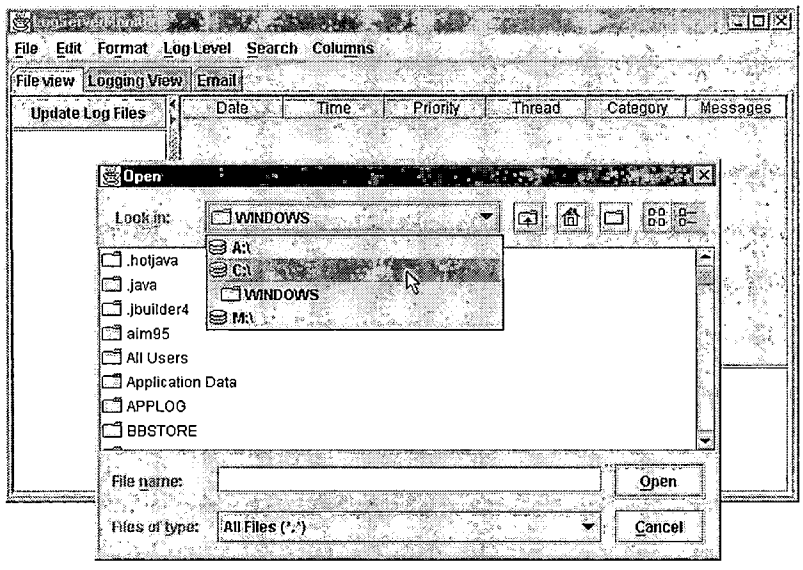


Figure 7. File Chooser Dialog (a)

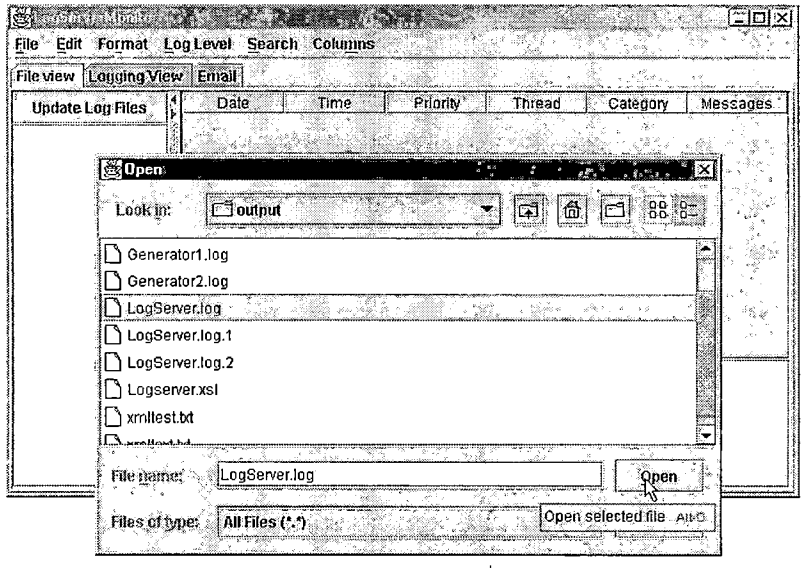


Figure 8. File Chooser Dialog (b)

The user can choose from the list area which log file to be displayed. When the user clicks on the "Update Log Files" button on the "File View" tab, the names of all log files and the time the files were created will be displayed in the list area on the left hand side (Figure 9).

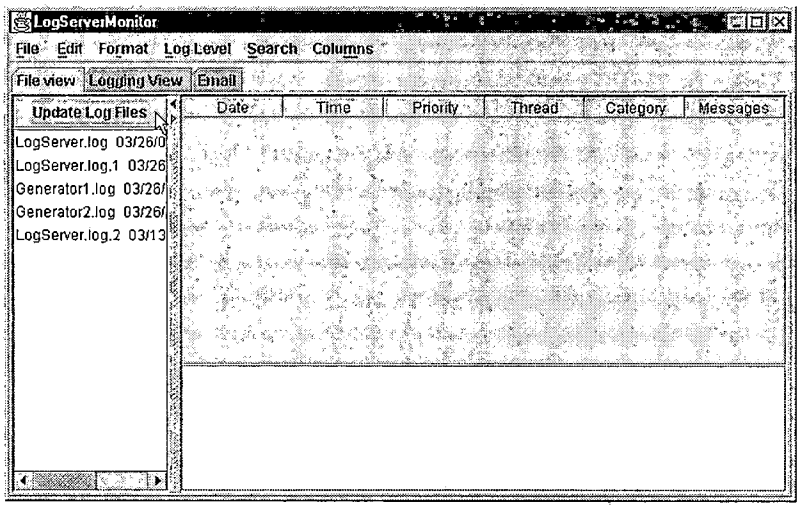


Figure 9. File List

When the user clicks on a file in the list area, the content of the selected file will be displayed in the table on the right-hand side (Figure 10).

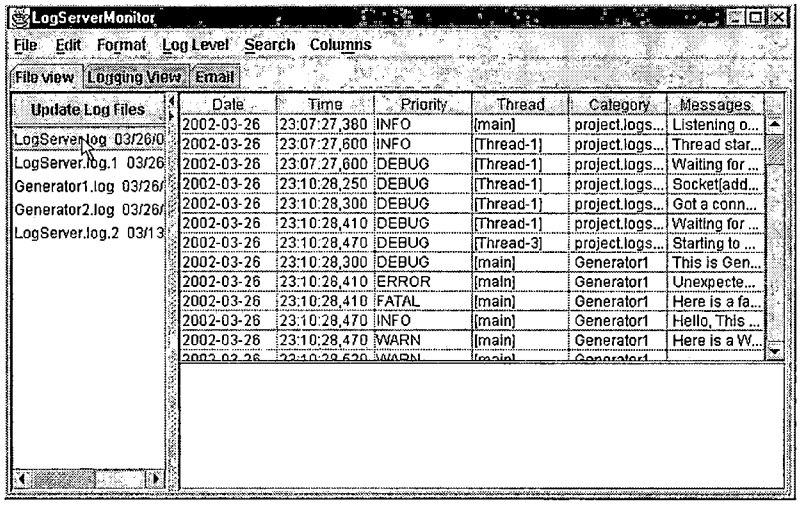


Figure 10. Select a File from the File List to Display

Sorting Messages by Table Columns. The messages in the table can be sorted by table columns. This can be done by clicking the table column header. For example, if you click the column header "Priority", then the messages in the table will be sorted by the content in the priority column (Figures 11 and 12).

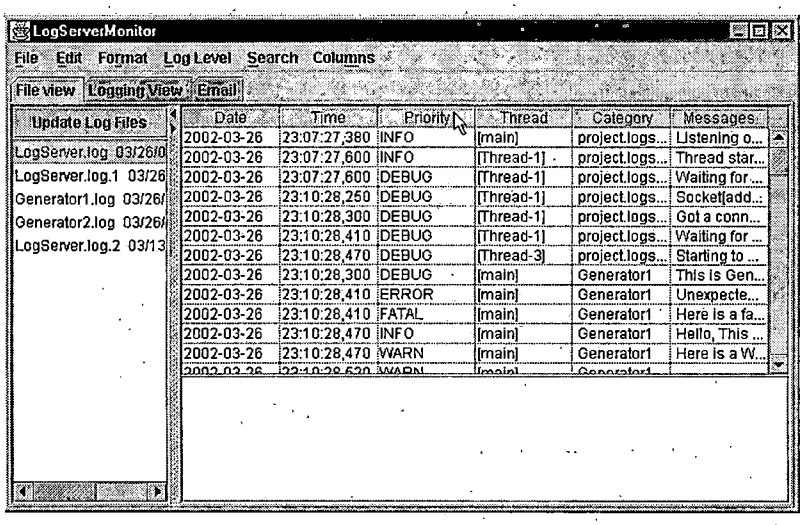


Figure 11. Sort Table (a)

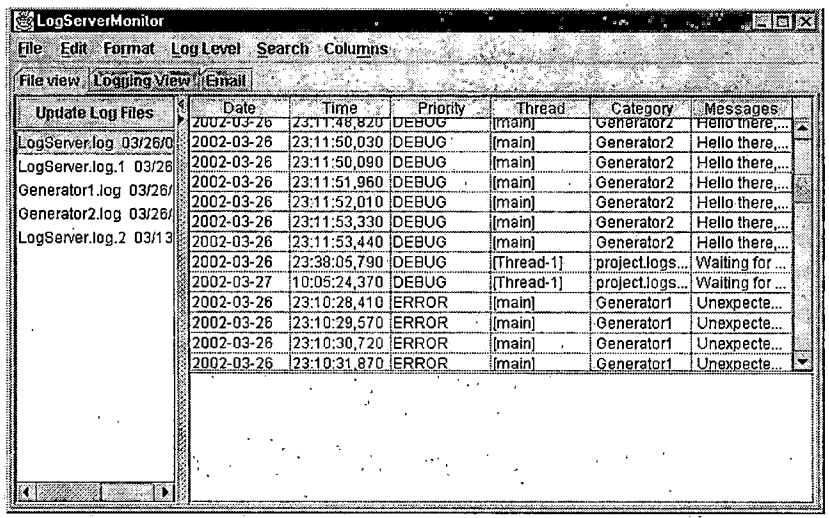


Figure 12. Sort Table (b)

Selecting a Message. The user can select a message from the table by clicking on this message. Then the selected message will be highlighted. Meanwhile, the detailed message will be displayed in the text area below the table (Figure 13).

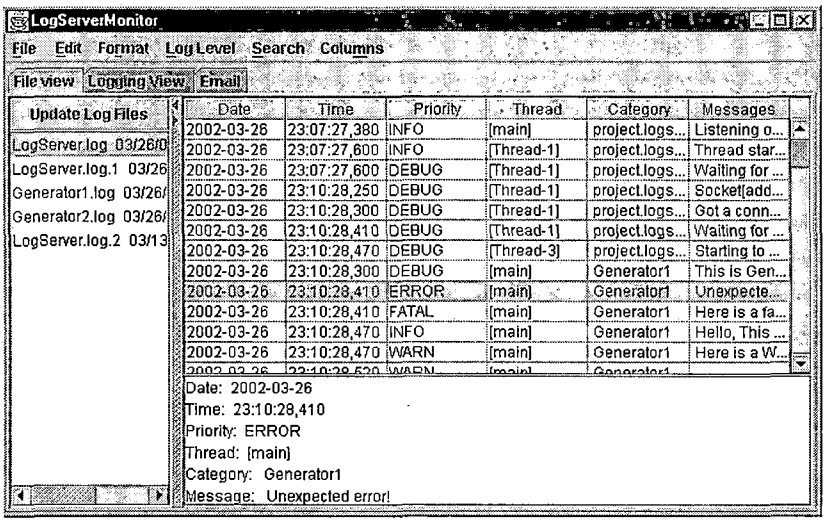


Figure 13. Select a Message

The "Edit" Menu. The "Edit" menu has "Copy" and "Paste" menu items. When the user clicks on the "Copy" menu item, the selected message from the table or the text area in the "Logging view" tab will be copied to the system clipboard. The copied message can be pasted in the text area on the "Email" tab or any other text editors such as Notepad by using their Paste function (Figures 14, 15, and 16).

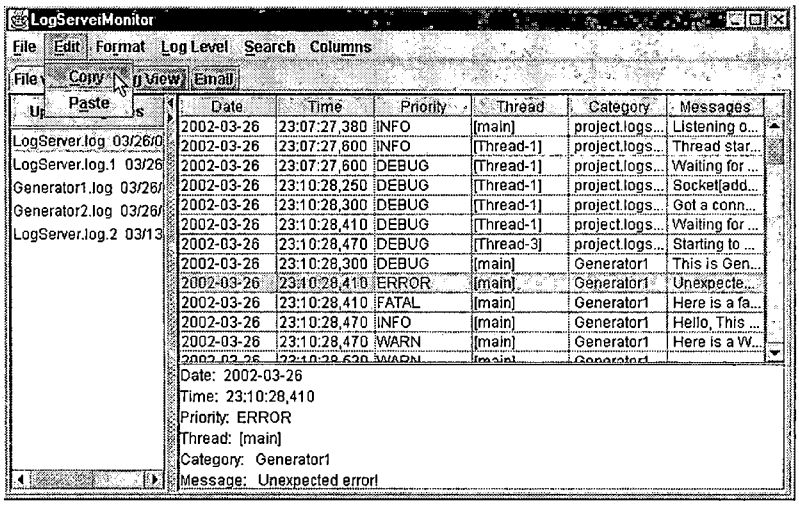


Figure 14. Copy

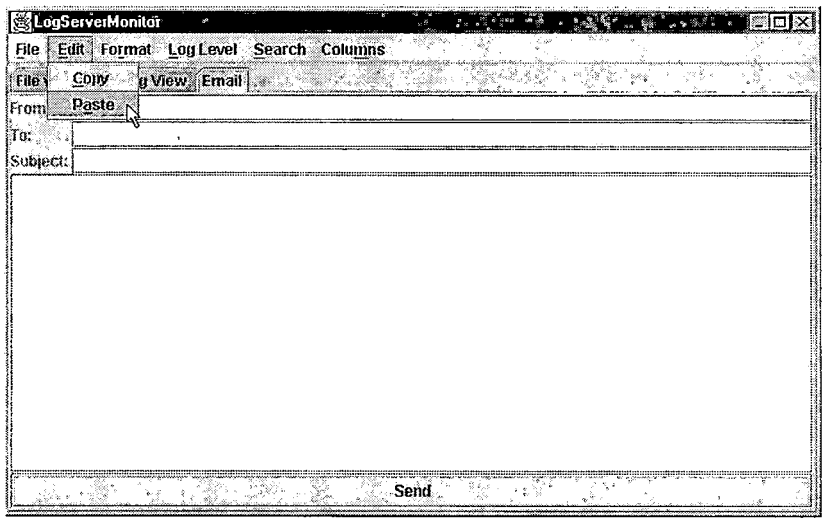


Figure 15. Paste (a)

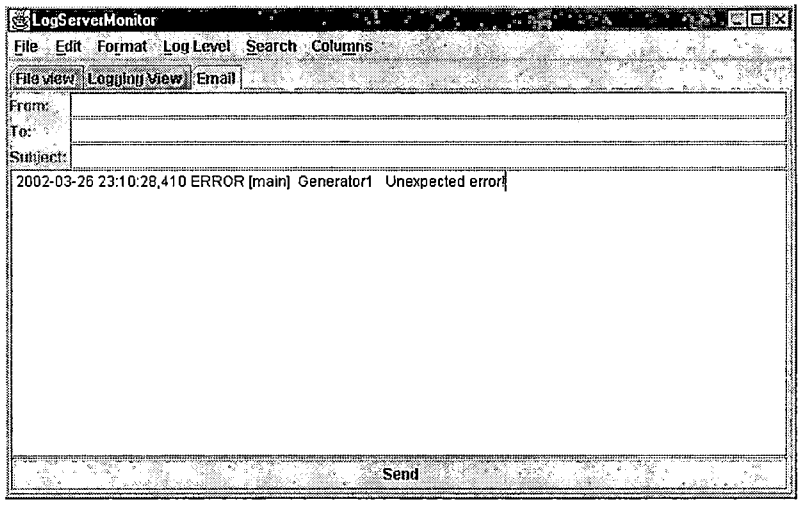


Figure 16. Paste (b)

The "Format" Menu. The "Format" menu has three menu items: "Font", "Color", and "Wrap."

When the user clicks on the "Font" menu item, a font dialog will pop up. The user can change font, size, and style for the text in the table (Figures 17 and 18).

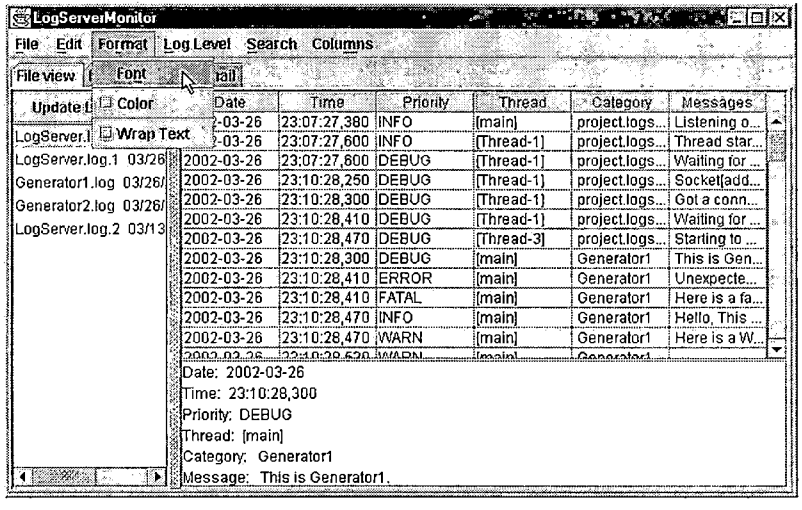


Figure 17. Change Font (a)

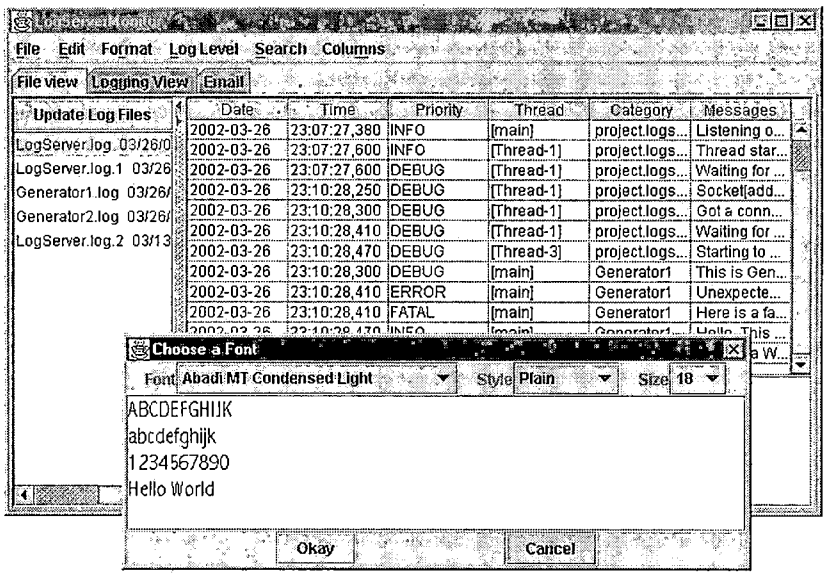


Figure 18. Change Font (b)

The user can also color code the text in the table by checking the "Color" menu item on the "Format" menu. The rows in the table will be colored according to the values in the "Priority" column. The color scheme is listed in the following table (Table 2, Figure 19).

Table 2. Color Scheme

Priority	Color
DEBUG	Black
INFO	Blue
WARN	Green
ERROR	Orange
FATAL	Red

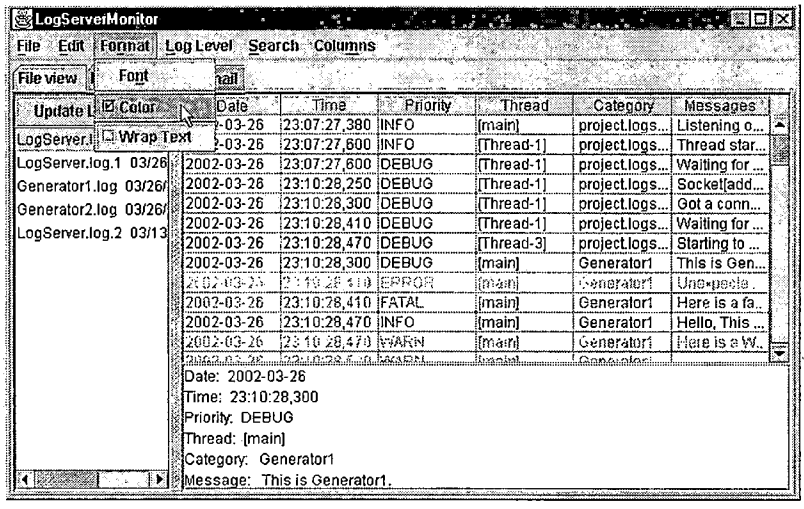


Figure 19. Color Coding Messages

If the user checks the "Wrap" JCheckBoxMenuItem on the "Format" menu, the text in the table will be wrapped. This is for the purpose of displaying the whole message when the messages are too long to be fitted on a single row in the table's cells (Figure 20).

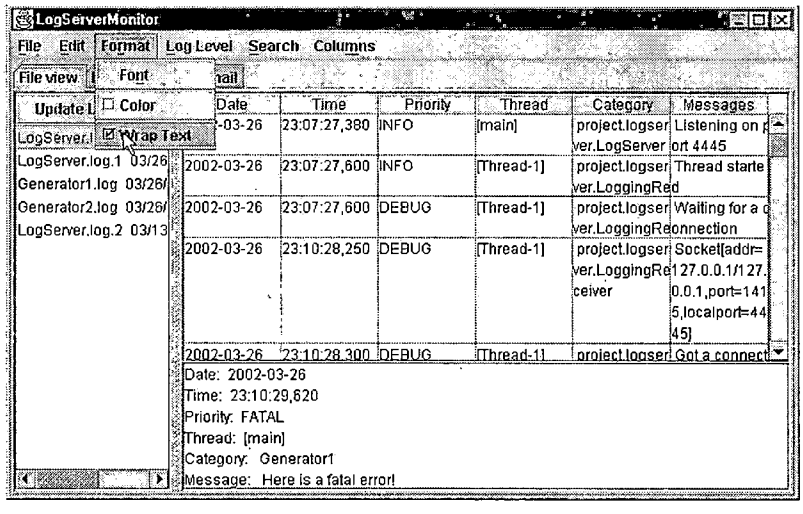


Figure 20. Wrap Text

The "Log Level" Menu. The "Log Level" menu has "DEBUG", "INFO", "WARN", "ERROR", and "FATAL" JcheckBoxMenuItems, corresponding to the types of values in the "Priority" column in the table. The table shows only the type of messages that are checked on the Log Level menu. For example, when the "DEBUG" menu item is unchecked, this type of message will not be displayed. Initially, all these check boxes are checked. The user can uncheck or recheck the check boxes to display desired type(s) of messages (Figures 21 and 22).

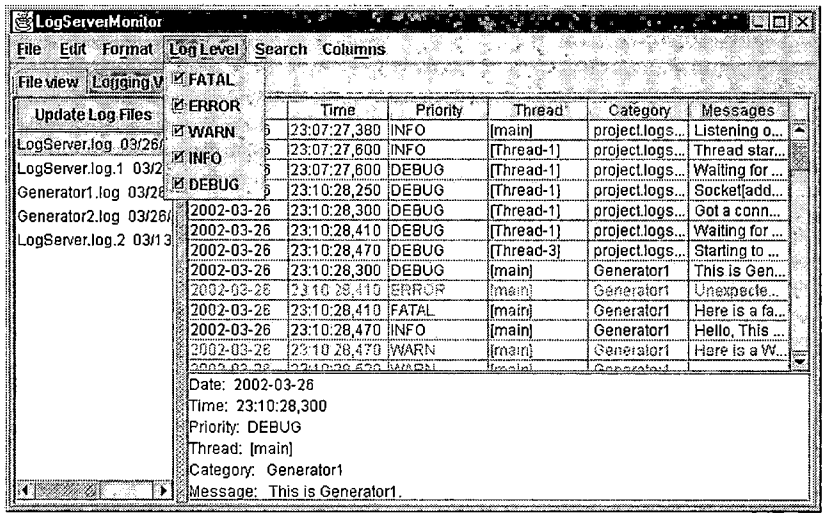


Figure 21. Log Level Filter (a)

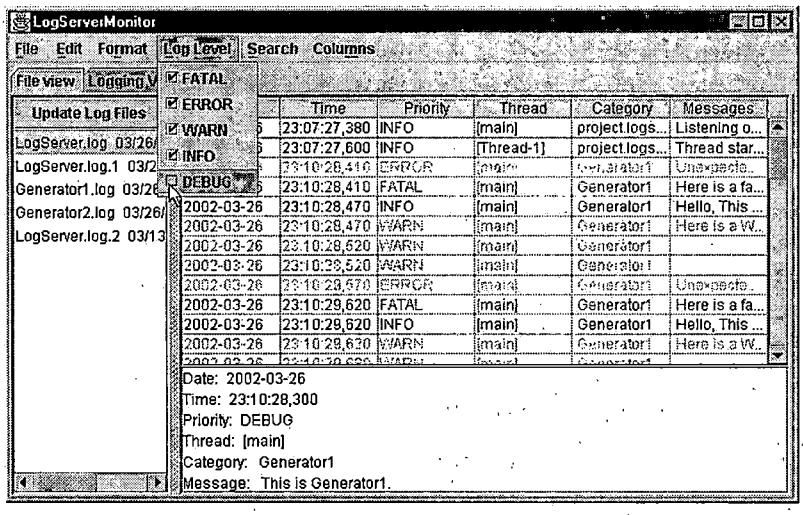


Figure 22. Log Level Filter (b)

The "Search" Menu. The "Search" menu has "Find" and "Find Next" menu items. From this menu, the user can search the messages in the table by keyword (Figure 23).

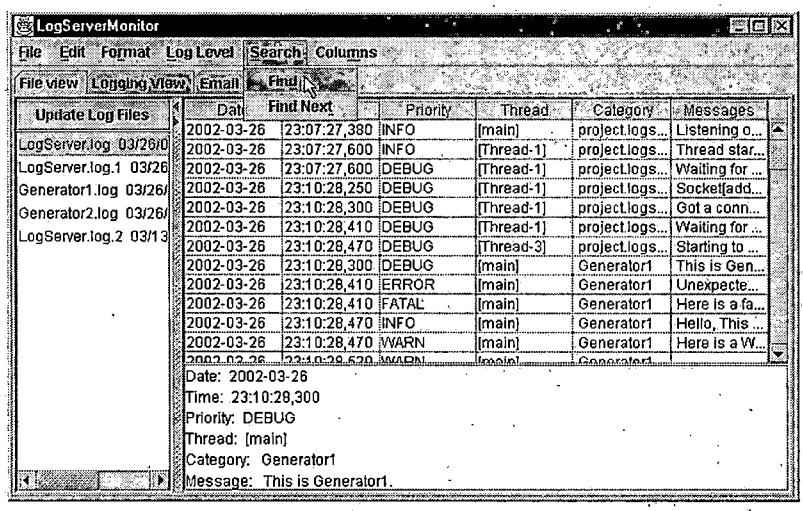


Figure 23. Search Menu

When the user clicks the "Find" menu item, a pop-up window will let the user type in the keyword to start the search (Figure 24).

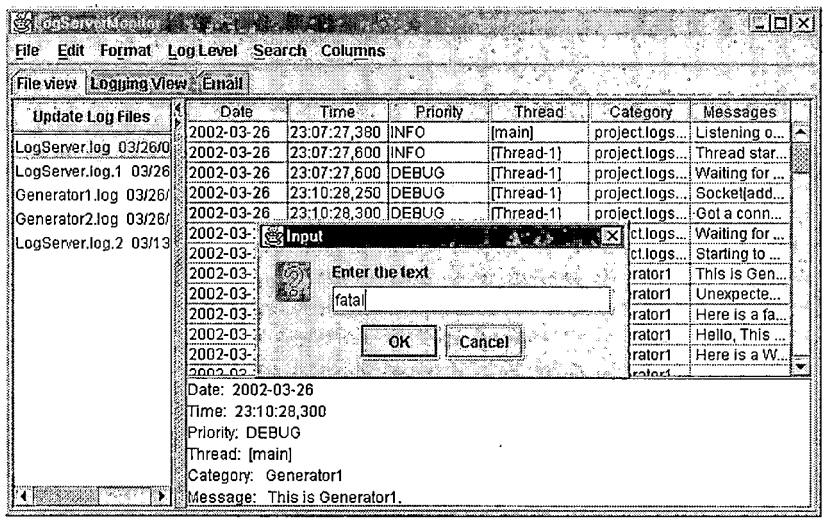


Figure 24. Find Message (a)

After the user types in the keyword and clicks the "OK" button, the message found will be highlighted in the table and the detailed message will be displayed in the text area below the table (Figures 25 and 26).

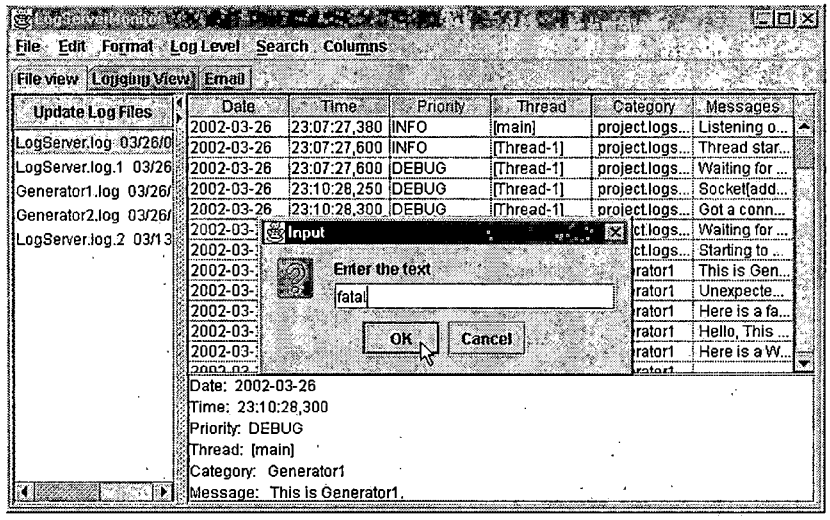


Figure 25. Find Message (b)

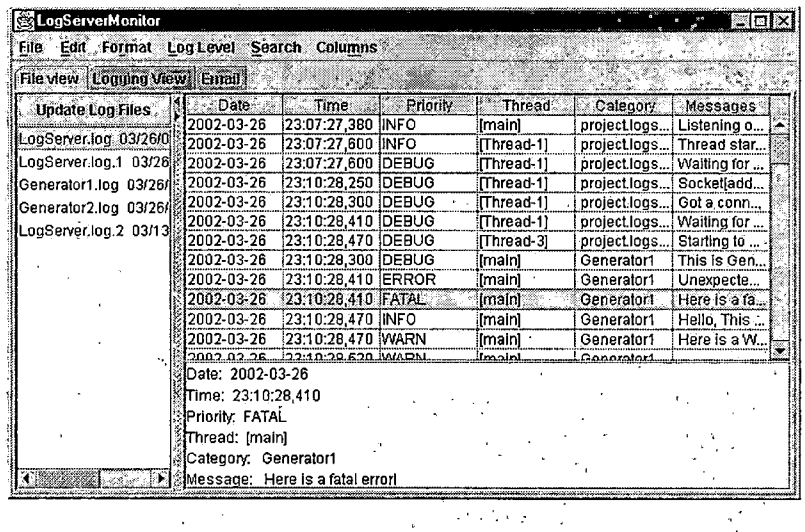


Figure 26. Find Message (c)

To find the next message containing the keyword, the user can click on the "Find Next" menu item. The next message found will be highlighted in the table and the detailed message will be displayed in the text area below the table (Figures 27 and 28).

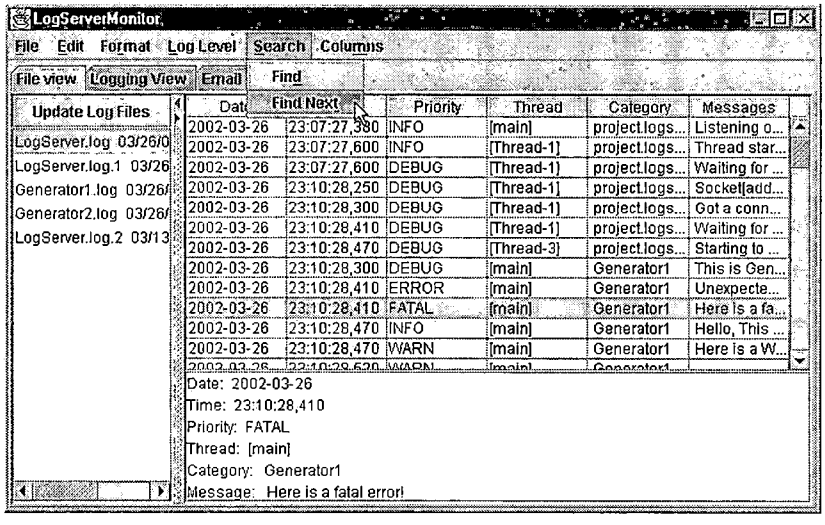


Figure 27. Find Next Message (a)

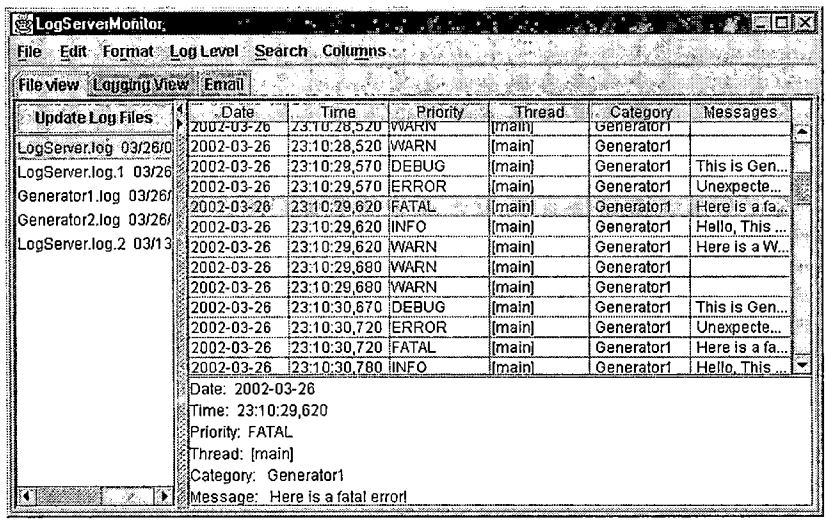


Figure 28 Find Next Message (b)

The "Columns" Menu. The "Columns" menu has "Date", "Time", "Priority", "Thread", "Category" and "Messages" JcheckBoxMenuItems corresponding to the table column names. All these checkboxes are checked by default. When these checkboxes are unchecked, the column with name matching the checkbox name will be turned off. For example, if the "Date" menu item is unchecked, the first column in the table will not be displayed (Figures 29 and 30).

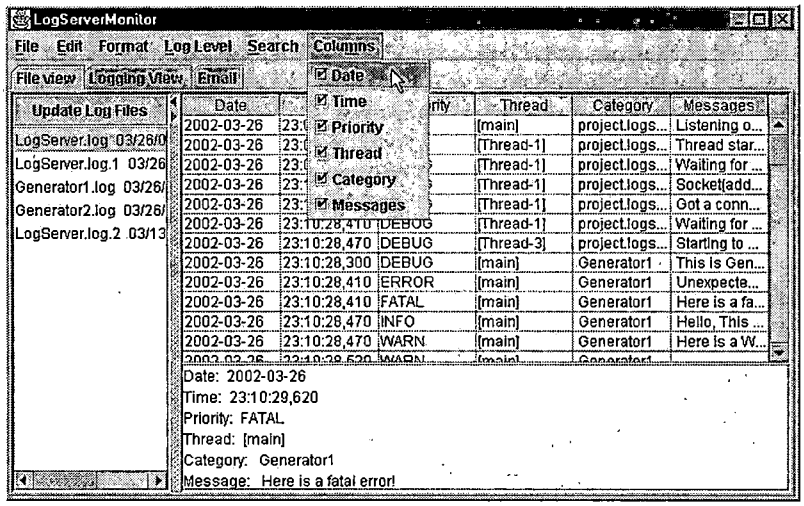


Figure 29. Hide/Show Columns (a)

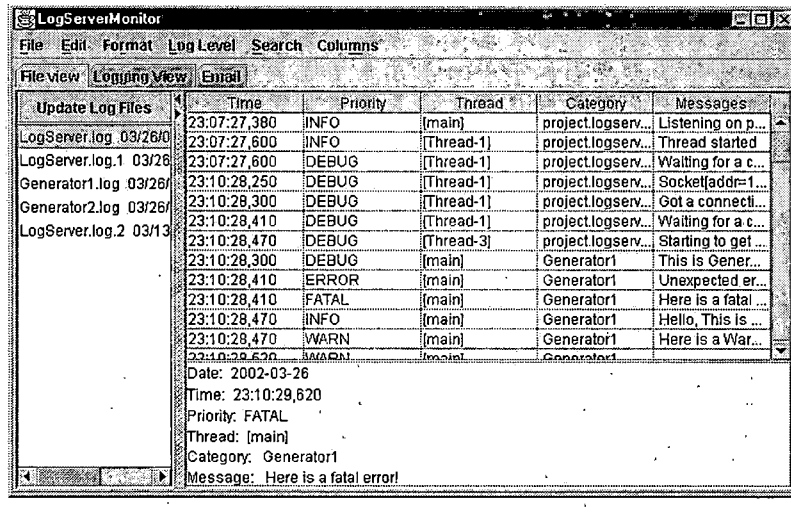


Figure 30. Hide/Show Columns (b)

The "Logging View" Tab

Displaying Clients Information. This tab is for dynamically displaying the log information. All of the messages logged from the clients are dynamically displayed in the text area as the messages are generated by a client program (Figure 31).

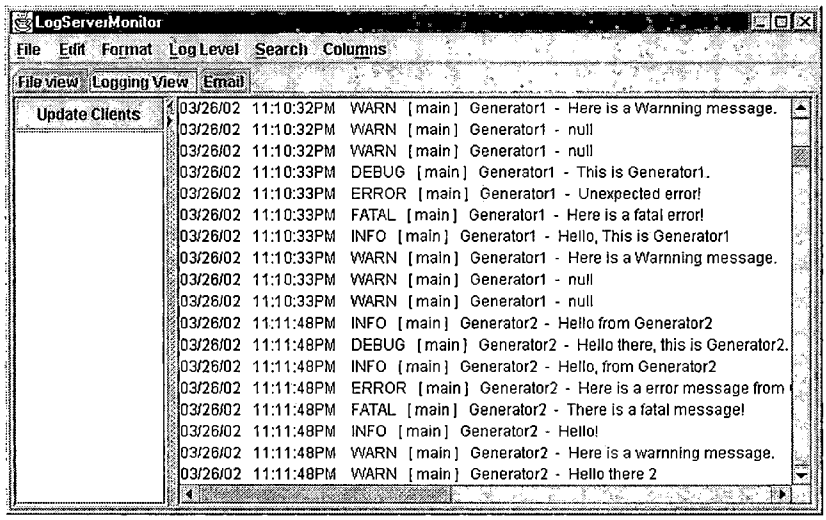


Figure 31. Logging Client Messages

List of Clients. All the names of the clients that are currently connected or have been connected with LogServer Monitor can be displayed in the list area on the left-hand side when the user clicks the "Update Clients" button (Figure 32).

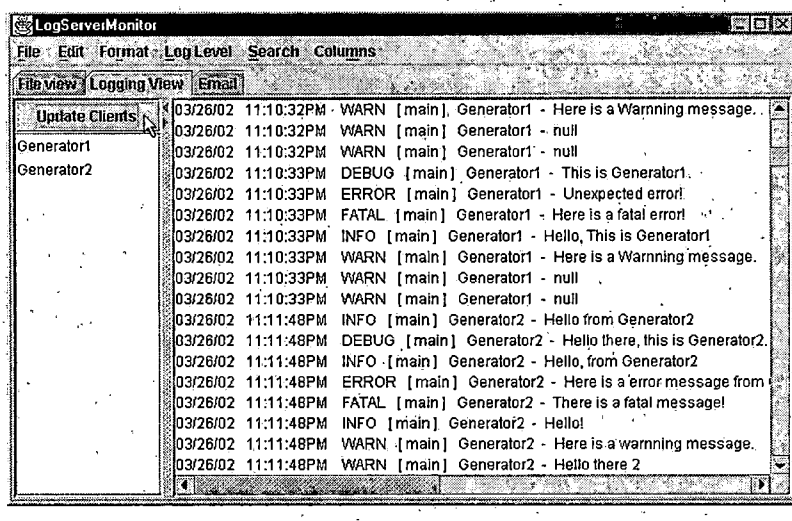


Figure 32. A List of Clients

Individual Client Window. The user can open the individual client window by clicking a client's name in the list (Figure 33).

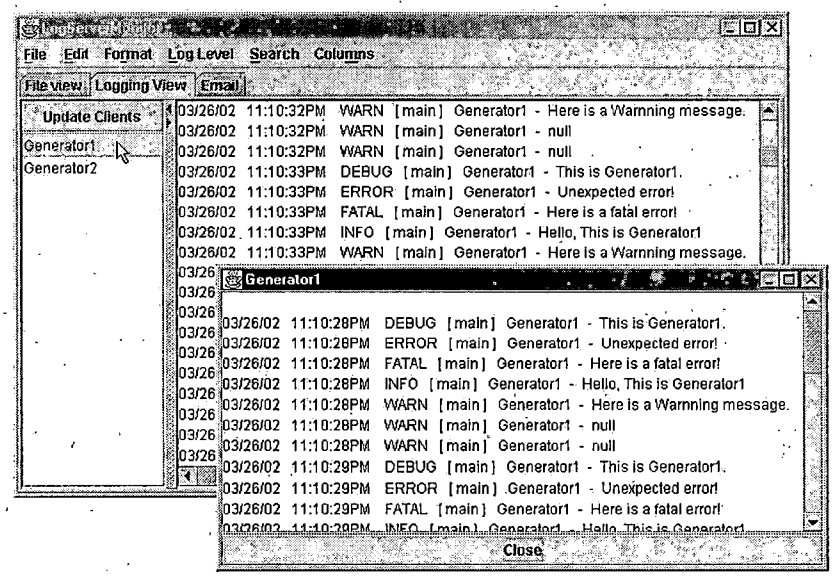


Figure 33. Individual Client Window

The "Email" Tab

The user can send emails to clients through this tab. When the user clicks the "Send" button, a message window will pop up reporting whether the email has been successfully sent or not (Figures 34 and 35).

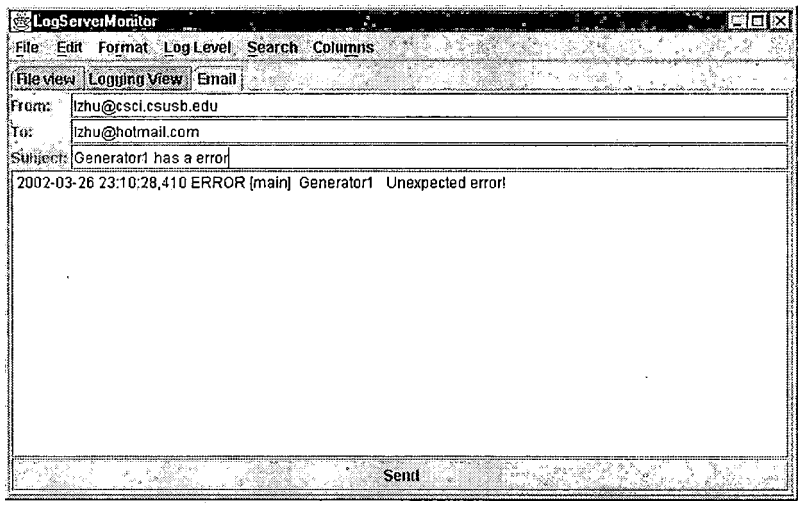


Figure 34. Send Email (a)

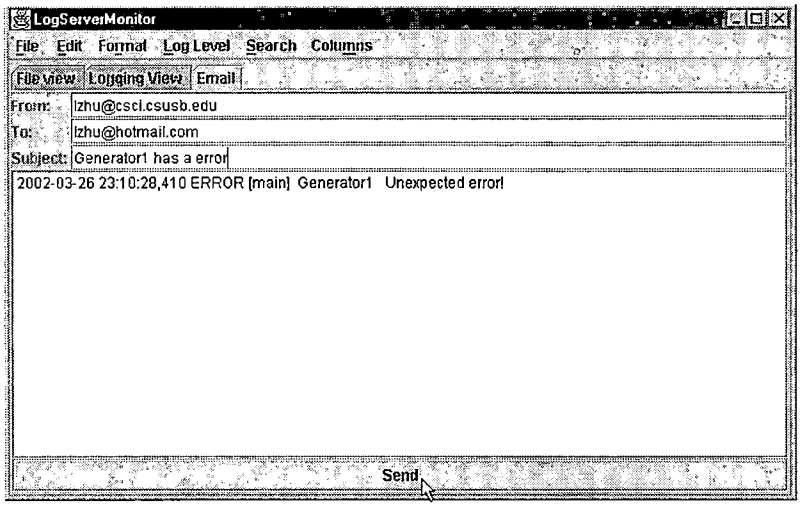


Figure 35. Send Email (b)

When the email has been sent, the user clicks the "OK" button in the message dialog. The "Email" tab will be refreshed and all the text areas will be cleared (Figures 36 and 37).

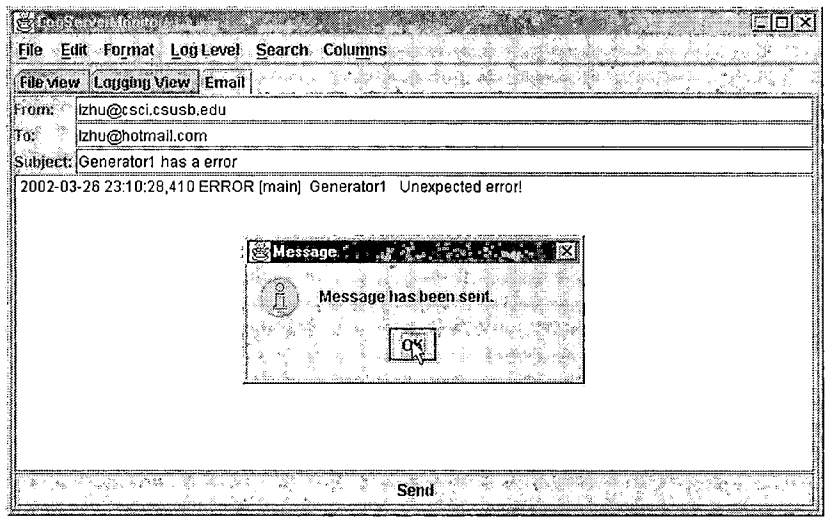


Figure 36. Send Email (c)

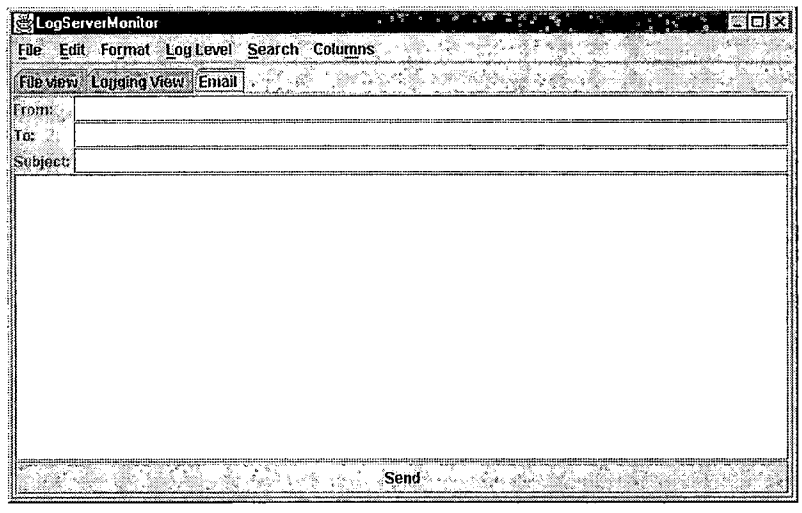


Figure 37. Send Email (d)

If the email has not been sent, a message window will pop up to report the error (Figure 38).

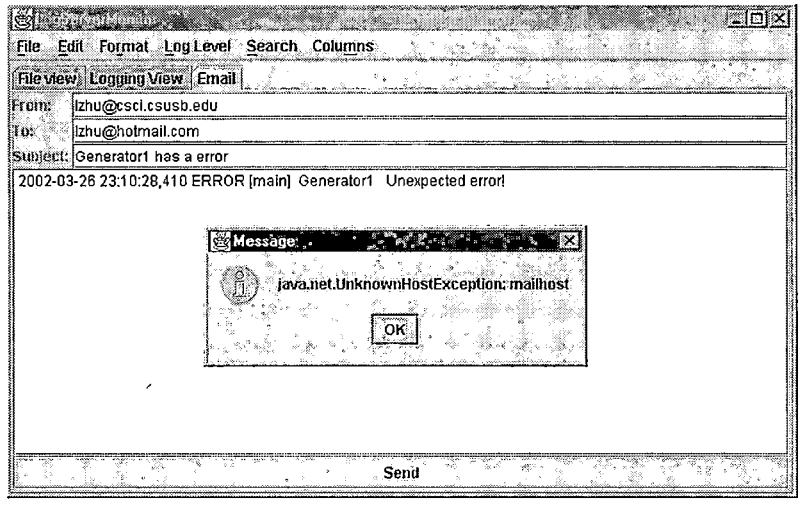


Figure 38. Send Email (e)

After the user clicks on the "OK" button on the window reporting the error condition, the text in all the text areas on the "Email" tab will still be there so that it can be resent when the error condition is cleared (Figures 39 and 40).

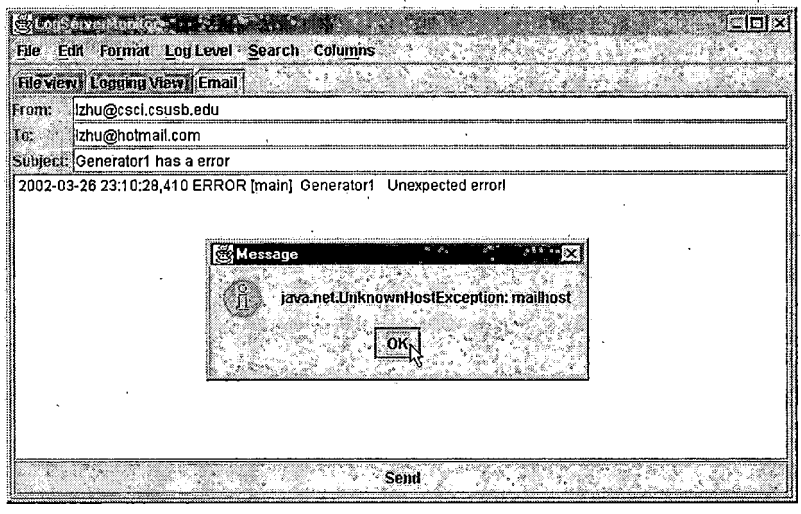


Figure 39. Send Email (f)

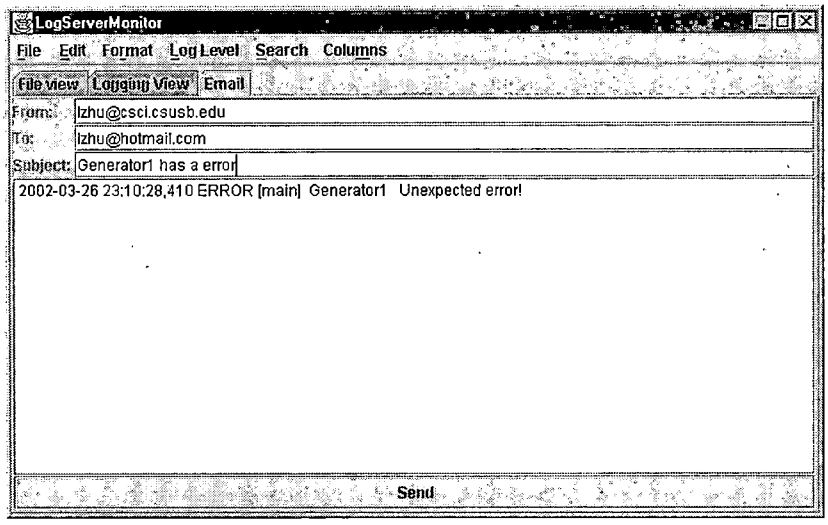


Figure 40. Send Email (g)

Class Descriptions

Packages Used in LogServer Monitor

As describe earlier, the LogServer Monitor uses the Java API, especially the Java "Swing" API, to create the graphic user interface (GUI). Specifically, the LogServer Monitor uses the following packages in the Java API:

- java.awt
- java.awt.datatransfer
- java.awt.event
- java.awt.geom
- java.awt.print
- java.net
- java.io

- java.util.StringTokenizer
- java.text.SampleDataFormat
- javax.swing
- javax.swing.table
- javax.swing.filechooser
- javax.swing.event
- java.util

The LogServer Monitor uses Log4J API, which includes:

- org.apache.log4j
- org.apache.log4j.Category
- org.apache.log4j.spi.LoggingEvent
- org.apache.log4j.Priority.

The New Classes Created for LogServer Monitor

The following classes have been newly created or adapted from other sources by the author for the LogServer Monitor:

- Data: stores the data contained in the table in a two-dimensional array.
- Message: defines the log message object according to the components of the log messages. For example, a message object may contain date, time, priority, thread, category, message, etc.

- MessageVector: reads the messages from a selected log file, stores the messages to a vector, and writes the log file to an XML file.
- MyTableModel: defines the number of rows and columns in the table.
- Properties: Parses the information that is in the properties file.
- TableMap[6]: implements the TableModelListener interface in the javax.swing.table package.
- TableSorter[6]: provides sorting functions for the table. The sorting algorithm used is stable, which means that it does not switch rows when its comparison function returns 0 to denote that they are equivalent.
- ColumnHandler: enables the user to toggle on or off columns. It implements the ItemListener interface in the java.awt.event package.
- LogLevelHandler: enables the user to pick which types (priorities) of messages to be displayed. It implements ItemListener interface in the java.awt.event package.
- MyTableCellRenderer: renders the cells in the table. This class sets color of the text in the table. It is

a subclass of DefaultTableCellRenderer in the javax.swing.table.package.

- TextAreaRenderer: performs text wrapping in the table cell. This class implements the TableCellRenderer interface in the javax.swing.table package. It assigns a JTextArea control to each table cell and sets the JTextArea's Wrap property to true.
- EventLogging: creates a new thread when a new client is connected to accept log messages from clients.
- FontChooser[3]: inherits the JDialog class in the javax.swing package, displays a dialog and allows the user to select a font in any style and size from the list of available fonts on the system.
- ItemChooser[3]: presents the choices of "Font", "Style", and "Size" in a ComboBox.
- LoggingReceiver: listens for client connections.
- LogServerMonitor: the main class, manages all other classes and interfaces.
- ProcFrame: creates a new frame for displaying individual client's logging messages.

Please see the appendix B for a more complete documentation of these classes such as their constructors

and methods. The sources for the adapted classes are also documented in Appendix B.

CHAPTER FOUR

SUMMARY AND CONCLUDING REMARKS

Summary

The LogServer Monitor is a system for displaying and monitoring log messages in a distributed system. It acts like a central server, which displays and monitors its clients' log information. The LogServer Monitor uses Java, Log4j, and XML technologies. The graphic user interface (GUI) of the LogServer monitor was designed to be user friendly. The log information from clients can be displayed dynamically in the LogServer Monitor GUI or saved on the server as log files which can be brought into the GUI. The user can send email to clients through the LogServer Monitor GUI to notify the client if there is a problem. The log files are also saved as XML files and an XML Style sheet (XSLT) file has been created for transforming the XML files to HTML on the fly.

Concluding Remarks

Logging is an important component of the software development and deployment. The logging captures the information such as configuration errors, performance bottlenecks, and bugs in the application. It can improve

the time of fixing a problem because the sooner an error detected, the cheaper it is to fix.

The LogServer Monitor provides a graphic user interface (GUI) to display and manage log information intuitively. The user can monitor how the application runs by examining the log information. The log information is also saved in log files to make it easy for the user to study at a later time.

These types of application are best written using Java programming language because it is easy to set up on almost any platform. Among many logging packages, Log4J is recommended because it is designed to be fast and flexible.

There are two limitations on the LogServer Monitor. First, the LogServer Monitor requires a fixed log file format. Otherwise, the log file cannot be displayed in the LogServer Monitor GUI properly. Secondly, the email function uses the mailto protocol. It requires that the local host be running an SMTP server. These limitations should be remedied in the future, improved, versions of this project.

APPENDIX A:

GLOSSARY OF TERMS

API	Application programming interface.
GUI	Graphic User Interface.
IRIX	An operating system developed by Silicon Graphic Ins.
Java	One of the most popular software development languages.
Log4j	A logging package for Java.
SMTP	Simple Mail Transport Protocol.
XML	Extensible Markup Language, a meta-markup language for text documents.
XSLT	Extensible Style sheet Language for Transformations, providing a standard way to transform XML files to a number of formats.

APPENDIX B:

CLASS DESCRIPIONS

project.logserver

Class Data

java.lang.Object

  |

  +--**project.logserver.Data**

public class **Data**

extends java.lang.Object

This class stores the data contained in the table in a two-dimensional array.

Constructor Detail

Data

public **Data**()

Data default constructor.

Method Detail

getData

public java.lang.Object[][] **getData**(java.lang.String fname, java.lang.String _dName)

Gets the data.

Parameters:

fname - the String representing the name of the input file to read from.

_dName - the String representing the
directory for the log files.

Returns:

an array of Object specifying the table
Data.

getsize

public int **getsize**()

Gets the array size.

Returns:

an integer specifying the array size.

getMessage

public java.util.Vector **getMessage**()

Gets the Vector for the message.

Returns:

a Vector.

updateMessage

public java.lang.Object[][] **updateMessage**(java.util.HashSet
hs)

Gets the updated messages.

Parameters:

hs - the HashSet that contains the priority.

Returns:

an array of Object specifying the table Data.

project.logserver

Class EventLogging

java.lang.Object

|

+--**project.logserver.EventLogging**

public class **EventLogging**

extends java.lang.Object

implements java.lang.Runnable

This class processes a client connection and receives log information from clients.

Method Detail

setFrame

public void **setFrame**(javax.swing.JTextArea cp)

Sets a JTextArea for displaying all clients information.

Parameters:

cp - the JTextArea.

run

public void **run**()

Listens for client connections.

Specified by:

run in interface java.lang.Runnable

project.logserver

Class FontChooser

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Window
                    |
                    +--java.awt.Dialog
                          |
                          +--javax.swing.JDialog
                                |
                                +--
```

project.logserver.FontChooser

public class **FontChooser**

extends javax.swing.JDialog

This is a JDialog subclass that allows the user to select a font, in any style and size, from the list of available fonts on the system. The source code is from the "Java Examples in a Nutshell" by David Flanagan.

Constructor Detail

FontChooser

public **FontChooser**(java.awt.Frame owner)

FontChooser constructor.

Parameters:

owner - the Frame

Method Detail

getSelectedFont

public java.awt.Font **getSelectedFont**()

Gets the user's selection. If the user used the "Cancel" button, this will return null.

Returns:

a Font.

getFontFamily

public java.lang.String **getFontFamily**()

Gets a font name.

Returns:

a String specifying the font name.

getFontStyle

public int **getFontStyle**()

Gets a font style.

Returns:

an integer specifying the font style.

getFontSize

public int **getFontSize**()

Gets a font size.

Returns:

an integer specifying the font size.

setFontFamily

public void **setFontFamily**(java.lang.String name)

Sets font.

Parameters:

name - the String representing a font name.

setFontStyle

public void **setFontStyle**(int style)

Sets font style.

Parameters:

style - the integer representing a font style.

setFontSize

public void **setFontSize**(int size)

Sets font size.

Parameters:

size - the integer representing a font size.

setSelectedFont

public void **setSelectedFont**(java.awt.Font font)

Sets selected font.

Parameters:

font - the Font

changeFont

protected void **changeFont**()

Changes the Font

isModal

public boolean **isModal**()

Override this inherited method.

Overrides:

isModal in class java.awt.Dialog

project.logserver

Class ItemChooser

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--javax.swing.JComponent
                    |
                    +--javax.swing.JPanel
                          |
                          +--project.logserver.ItemChooser
```

public class **ItemChooser**

extends javax.swing.JPanel

This class presents the choices of "Font", "Style", and "Size" in ComboBox. This code is adapted from the "JAVA Examples in the Nutshell" by David Flanagan.

Constructor Detail

ItemChooser

public **ItemChooser**(java.lang.String name,
java.lang.String[] labels,

java.lang.Object[] values,

int defaultSelection)

ItemChooser constructor.

Parameters:

name - the String representing a choice name.

labels - the array of String for each choice option.

values - the array of Objects associated with each option.

defaultSelect - the integer that represents a Font.

Method Detail

getName

public java.lang.String **getName**()

Gets choice name.

Overrides:

getName in class java.awt.Component

Returns:

a String representing the choice name.

getLabels

public java.lang.String[] **getLabels**()

Gets array of String for each choice option.

Returns:

an array of String.

getValues

public java.lang.Object[] **getValues**()

Gets array of an object associated with each option.

Returns:

an array of Object.

getSelectedIndex

public int **getSelectedIndex**()

Gets selected index.

Returns:

an integer.

getSelectedValue

public java.lang.Object **getSelectedValue**()

Gets selected value.

Returns:

an Object.

setSelectedIndex

public void **setSelectedIndex**(int selection)

Sets selected index.

Parameters:

selection - the integer representing the selected value.

select

protected void **select**(int selection)

Stores the new selected index and fires events to any registered listeners.

Parameters:

selection - the integer.

addItemChooserListener

public void **addItemChooserListener**(ItemChooser.Listener listener)

Registers the event listener.

Parameters:

listener - the ItemChooser.Listener.

removeItemChooserListener

public void **removeItemChooserListener**(ItemChooser.Listener listener)

Deregisters the event listener.

Parameters:

listener - the ItemChooser.Listener.

project.logserver

Class ItemChooser.Event

java.lang.Object

  |

  +--java.util.EventObject

        |

        +--project.logserver.ItemChooser.Event

Enclosing class:

ItemChooser

public static class **ItemChooser.Event**

extends java.util.EventObject

This inner class defines the event type generated by ItemChooser.

Constructor Detail

ItemChooser.Event

public **ItemChooser.Event**(ItemChooser source,

int selectedIndex,

java.lang.Object

selectedValue)

Event constructor.

Parameters:

source - the ItemChooser object.

selectedIndex - the integer for the index.

selectedValue - the object for selected value.

Method Detail

getItemChooser

public ItemChooser **getItemChooser**()

Gets ItemChooser.

Returns:

an ItemChooser.

getSelectedIndex

public int **getSelectedIndex**()

Gets selected index.

Returns:

an integer.

getSelectedValue

public java.lang.Object **getSelectedValue**()

Gets selected value.

Returns:

an Object representing the selected value.

project.logserver

Interface ItemChooser.Listener

Enclosing class:

ItemChooser

public static interface **ItemChooser.Listener**

extends java.util.EventListener

This interface must be implementd by any object that wants to be notified when the current selection in an ItemChooser component changes.

Method Detail

itemChosen

public void **itemChosen**(ItemChooser.Event e)

project.logserver

Class LoggingReceiver

java.lang.Object

  |

  +--java.lang.Thread

        |

        +--**project.logserver.LoggingReceiver**

public class **LoggingReceiver**

extends java.lang.Thread

This class processes connections from the clients.

Method Detail

run

public void **run**()

Listens for client connections.

Overrides:

run in class java.lang.Thread

getHashtable

public java.util.Hashtable **getHashtable**()

Gets the hashtable that contains the individual clients window.

Returns:

a Hashtable.

getClientList

public java.util.Vector **getClientList**()

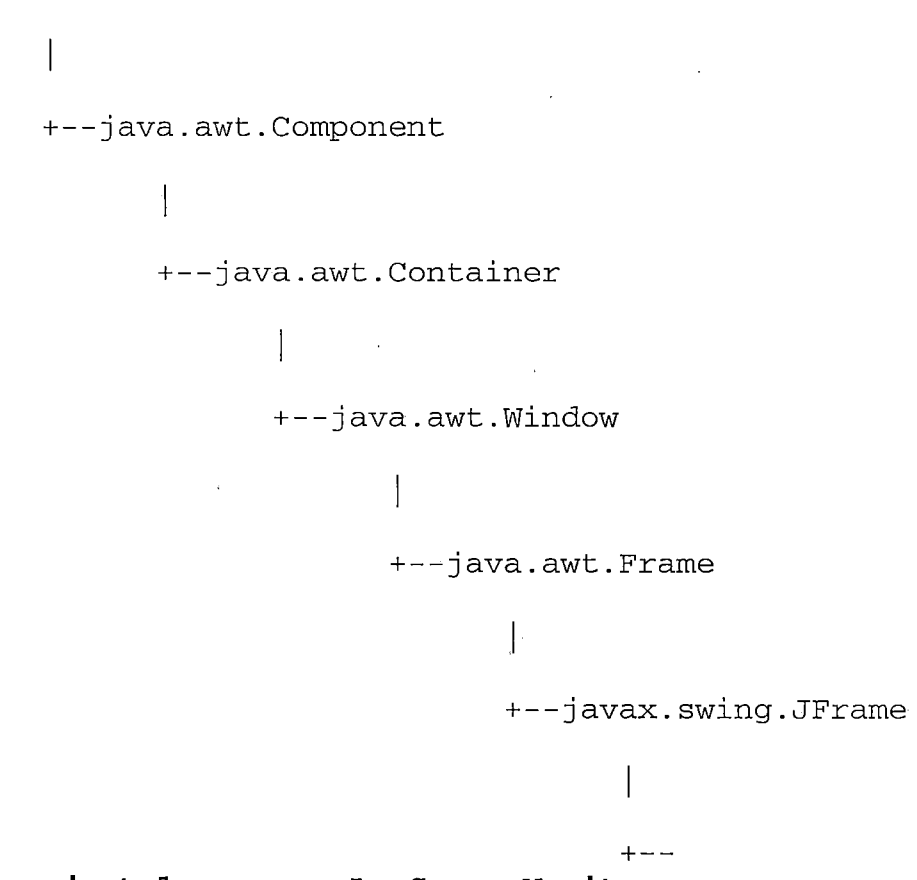
Gets the clients list.

Returns:

a Vector.

project.logserver

Class LogServerMonitor



public class **LogServerMonitor**

extends javax.swing.JFrame

implements java.awt.print.Printable

This is the main class for the LogServer Monitor.

Constructor Detail

LogServerMonitor

public **LogServerMonitor**()

LogServermonitor constructor.

Method Detail

search

public void **search**(int index,

java.lang.String input)

Searchs the message in the table.

Parameters:

index - the index of the row in the table on which to start the search.

getPosition

public int **getPosition**()

Gets the index of the row on which the message was found.

Returns:

an integer.

getFileList

public java.util.Vector **getFileList**()

Gets the log file list.

Returns:

a Vector.

createTablePane

protected javax.swing.JSplitPane **createTablePane**()

Creates a tab with a JTable, JList, JButton, and JTextArea.

createClientPane

protected javax.swing.JSplitPane **createClientPane**()

Creates a tab with a JTextArea, JButton, and JList.

createEmailPane

protected javax.swing.JPanel **createEmailPane**()

Creates the Email tab.

sendMail

public void **sendMail**()

Sends the email through the URL connection.

copy

public void **copy**()

Copys the selected text in the JTable or in the JTextArea to the system clipboard.

paste

public void **paste**()

Pastes the text from the system clipboard to the Email tab.

Print[6]

public int **print**(java.awt.Graphics g,
java.awt.print.PageFormat pageFormat,
int pageIndex)
throws java.awt.print.PrinterException

Print function.

Specified by:

print in interface java.awt.print.Printable

Parameters:

g - the Graphics.

pageFormat - the PageFormat.

pageIndex - the page index.

Returns:

an integer.

GetPageInfo [6]

public void **getPageInfo**(java.awt.Graphics g,
java.awt.print.PageFormat
pageFormat)

Gets page information.

Parameters:

g - the Graphics.

pageFormat - the PageFormat.

PrintTablePart [6]

public void **printTablePart**(java.awt.Graphics2D g2,
java.awt.print.PageFormat
pageFormat,
int rowIndex,
int columnIndex)

Prints the table.

Parameters:

g2 - the Graphic2D.

pageFormat - the PageFormat.

rowIndex - the table row index.

columnIndex - the table column index.

main

public static void **main**(java.lang.String[] args)

The main method.

Parameters:

args[0] - the property file.

project.logserver

Class LogServerMonitor.ColumnHandler

java.lang.Object

  |

  +--**project.logserver.LogServerMonitor.ColumnHandler**

Enclosing class:

LogServerMonitor

public class **LogServerMonitor.ColumnHandler**

extends java.lang.Object

implements java.awt.event.ItemListener

This class enables the user to toggle on or off columns.

Constructor Detail

LogServerMonitor.ColumnHandler

public **LogServerMonitor.ColumnHandler**()

Method Detail

itemStateChanged

public void **itemStateChanged**(java.awt.event.ItemEvent e)

This method is called if the user selects a JCheckBoxMenuItem in the columnMenu.

Specified by:

itemStateChanged in interface

java.awt.event.ItemListener

Parameters:

e - the ItemEvent.

project.logserver

Class LogServerMonitor.LogLevelHandler

java.lang.Object

|

+--**project.logserver.LogServerMonitor.LogLevelHandler**

Enclosing class:

LogServerMonitor

public class **LogServerMonitor.LogLevelHandler**

extends java.lang.Object

implements java.awt.event.ItemListener

This class enables the user to pick which types (priorities) of messages to be displayed.

Constructor Detail

LogServerMonitor.LogLevelHandler

public **LogServerMonitor.LogLevelHandler**()

Method Detail

itemStateChanged

public void **itemStateChanged**(java.awt.event.ItemEvent e)

This method is called if the user selects a JCheckBoxMenuItem in the logMenu.

Specified by:

itemStateChanged in interface

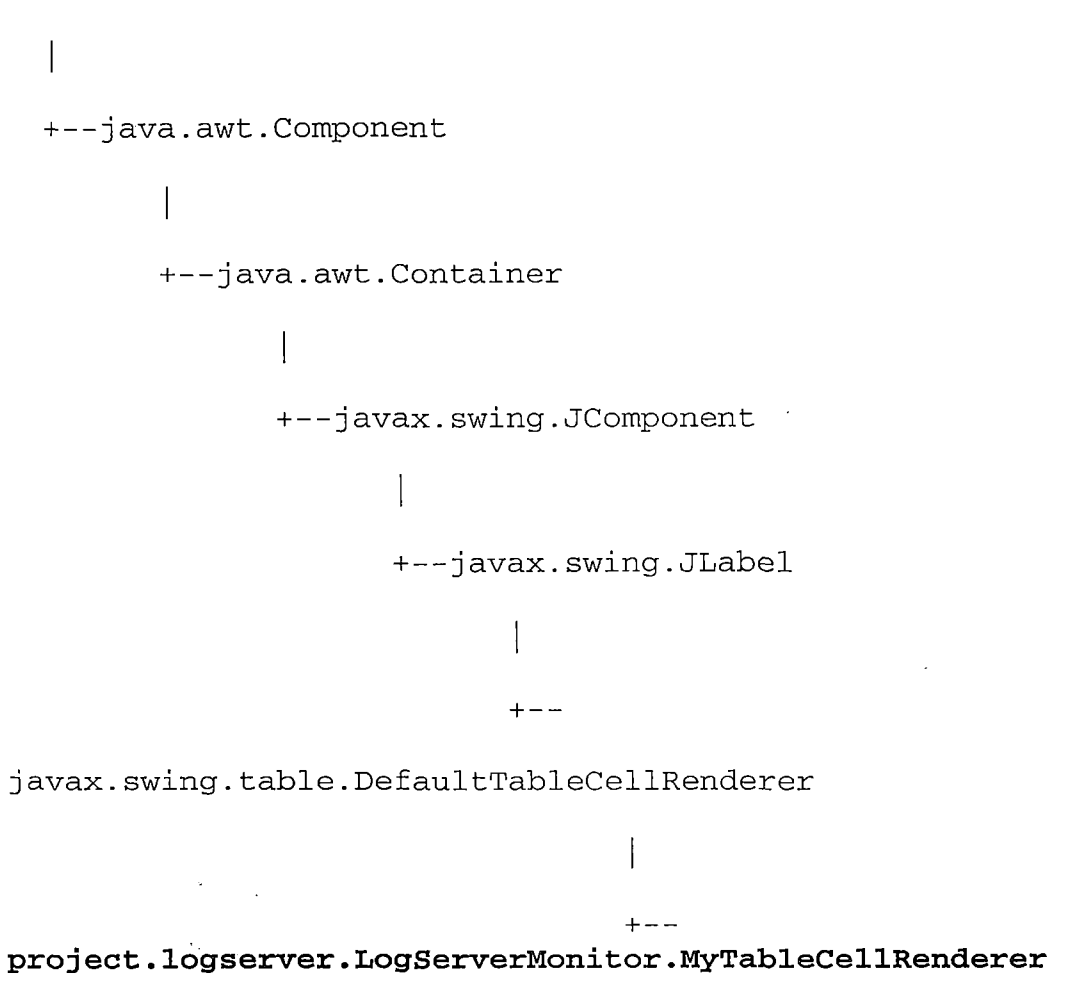
java.awt.event.ItemListener

Parameters:

e - the ItemEvent.

project.logserver

Class LogServerMonitor.MyTableCellRenderer



Enclosing class:

LogServerMonitor

public class **LogServerMonitor.MyTableCellRenderer**

extends javax.swing.table.DefaultTableCellRenderer

This class renders the cells in the table and sets the color of the text in the table.

Constructor Detail

LogServerMonitor.MyTableCellRenderer

public **LogServerMonitor.MyTableCellRenderer**()

Method Detail

getTableCellRendererComponent

public java.awt.Component

getTableCellRendererComponent(javax.swing.JTable table, java.lang.Object value, boolean isSelected, boolean hasFocus, int row, int column)

Returns the table cell renderer.

Overrides:

getTableCellRendererComponent in class javax.swing.table.DefaultTableCellRenderer

Parameters:

table - the JTable.

value - the value to assign to the cell at [row, column].

isSelected - true if cell is selected.

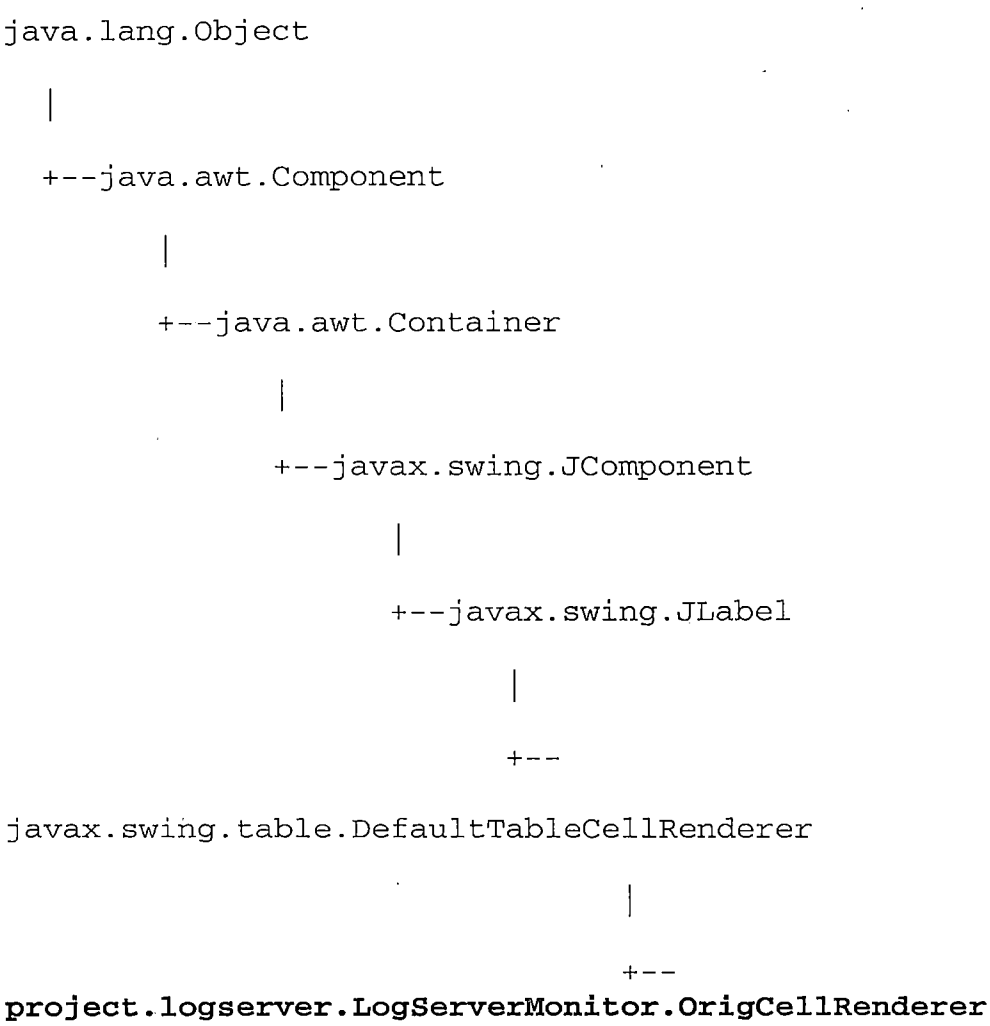
hasFocus - true if cell has focus.

row - the row of the cell to render.

column - the column of the cell to render.

project.logserver

Class LogServerMonitor.OrigCellRenderer



Enclosing class:

LogServerMonitor

public class **LogServerMonitor.OrigCellRenderer**

extends javax.swing.table.DefaultTableCellRenderer

This class renders the cells in the table.

Constructor Detail

LogServerMonitor.OrigCellRenderer

public **LogServerMonitor.OrigCellRenderer**()

Method Detail

getTableCellRendererComponent

public java.awt.Component

getTableCellRendererComponent(javax.swing.JTable table,

java.lang.Object value, boolean isSelected,

boolean hasFocus, int row, int column)

Returns the table cell renderer.

Overrides:

getTableCellRendererComponent in class

javax.swing.table.DefaultTableCellRenderer

Parameters:

table - the JTable.

value - the value to assign to the cell at

[row, column].

isSelected - true if cell is selected.

hasFocus - true if cell has focus.

row - the row of the cell to render.

column - the column of the cell to render.

project.logserver

Class LogServerMonitor.TextAreaRenderer

Enclosing class:

LogServerMonitor

public class **LogServerMonitor.TextAreaRenderer**

extends javax.swing.JTextArea

implements javax.swing.table.TableCellRenderer

This calss performs text wrapping in the table cell.

Constructor Detail

LogServerMonitor.TextAreaRenderer

public **LogServerMonitor.TextAreaRenderer**()

Method Detail

getTableCellRendererComponent

public java.awt.Component

getTableCellRendererComponent(javax.swing.JTable table, java.lang.Object value, boolean isSelected, boolean hasFocus, int row, int column)

Returns the table cell renderer.

Specified by:

getTableCellRendererComponent in interface javax.swing.table.TableCellRenderer

Parameters:

table - the JTable.

value - the value to assign to the cell at [row, column].

isSelected - true if cell is selected.

hasFocus - true if cell has focus.

row - the row of the cell to render.

column - the column of the cell to render.

project.logserver

Class Message

java.lang.Object

  |

  +--project.logserver.Message

public class Message

extends java.lang.Object

This class defines the log message object according to the components of the log messages. For example, a message object may contain date, time, priority, thread, category, message, etc.

Constructor Detail

Message

public **Message**()

    Message default constructor.

Message

public **Message**(java.lang.String _date,

                  java.lang.String _time,

                  java.lang.String _priority,

                  java.lang.String _tName,

                  java.lang.String _cName,

java.lang.String m)

Message constructor.

Parameters:

_date - the String representing the date.

_time - the String representing the time.

_proority - the String representing the priority.

_tName - the String representing the thread.

_cName - the String representing the category.

m - the String representing the message.

Method Detail

setDate

public void **setDate**(java.lang.String _date)

Sets the date.

Parameters:

_date - the String representing the date.

setTime

public void **setTime**(java.lang.String _time)

Sets the time.

Parameters:

_time - the String representing the time.

setPriority

public void **setPriority**(java.lang.String _priority)

Sets the priority.

Parameters:

_priority - the String representing the priority.

setThread

public void **setThread**(java.lang.String _tName)

Sets the thread.

Parameters:

_tName - the String representing the thread.

setCategory

public void **setCategory**(java.lang.String _cName)

Sets the category.

Parameters:

_cName - the String representing the category.

setMessage

public void **setMessage**(java.lang.String m)

Sets the message.

Parameters:

m - the String representing the message.

getDate

public java.lang.String getDate()

Gets the date.

Returns:

a String representing the date.

getTime

public java.lang.String **getTime**()

Gets the time.

Returns:

a String representing the time.

getPriority

public java.lang.String **getPriority**()

Gets the priority.

Returns:

a String representing the priority.

getThread

public java.lang.String **getThread**()

Gets the thread.

Returns:

a String representing the thread.

getCategory

public java.lang.String **getCategory**()

Gets the category.

Returns:

a String representing the category.

getMessage

public java.lang.String **getMessage**()

Gets the message.

Returns:

a String representing the message.

project.logserver

Class MessageVector

java.lang.Object

|

+--**project.logserver.MessageVector**

public class MessageVector

extends java.lang.Object

This class reads the messages from a selected log file, stores the messages to a vector, and writes the log file to an XML file.

Constructor Detail

MessageVector

public **MessageVector**()

MessageVector default constructor.

Method Detail

readFile

public void **readFile**(java.lang.String file_Name,
java.lang.String dir_Name)

Reads log information from a log file, puts the information in a Vector, and writes the log information to an XML file.

Parameters:

file_Name - the String representing the name of the input file to read from.

dir_Name - the String representing the directory for the log files.

getVector

public java.util.Vector **getVector**()

Returns a Vector that stores log messages.

Returns:

a Vector.

project.logserver

Class MyTableModel

java.lang.Object

  |

  +--javax.swing.table.AbstractTableModel

        |

        +--**project.logserver.MyTableModel**

public class **MyTableModel**

extends javax.swing.table.AbstractTableModel

This class defines the table model.

Constructor Detail

MyTableModel

public **MyTableModel**()

    The MyTablemodel default construtor.

Method Detail

setTable

public void **setTable**(java.lang.String fileName,

                    java.lang.String dName)

    Sets the table data.

    Parameters:

        fileName - the String representing the name

of the input file to read from.

dName - the String representing the

directory for the log files.

resetTable

public void **resetTable**(java.util.HashSet hs)

Resets the table data.

Parameters:

hs - the HashSet.

getColumnCount

public int **getColumnCount**()

Gets the number of columns in the model.

Overrides:

getColumnCount in class

javax.swing.table.AbstractTableModel

Returns:

an integer.

getRowCount

public int **getRowCount**()

Gets the number of rows in the model.

Overrides:

getRowCount in class

javax.swing.table.AbstractTableModel

Returns:

an integer.

getColumnName

public java.lang.String **getColumnName**(int col)

Gets the name of the column at col.

Overrides:

getColumnName in class

javax.swing.table.AbstractTableModel

Parameters:

col - the index of the column.

Returns:

a String representing the column name.

getValueAt

public java.lang.Object **getValueAt**(int row,

int col)

Gets the value for the cell at col and row.

Overrides:

getValueAt in class

javax.swing.table.AbstractTableModel

Parameters:

row - the row whose value is to be queried.

col - the column whose value is to be queried.

Returns:

a value Object at the specified cell.

getColumnClass

public java.lang.Class **getColumnClass**(int c)

Gets the most specific superclass for all the cell values in the column.

Overrides:

getColumnClass in class javax.swing.table.AbstractTableModel

Parameters:

c - the index of the column.

Returns:

a common ancestorClass of the object values in the model.

isCellEditable

public boolean **isCellEditable**(int row,
int col)

Returns true if the cell is editable, false otherwise.

Overrides:

isCellEditable in class

javax.swing.table.AbstractTableModel

Parameters:

row - the row whose value to be queried.

col - the column whose value to be queried.

Returns:

true if the cell is editable.

setValueAt

public void **setValueAt**(java.lang.Object value,

int row,

int col)

Sets the value in the cell at col and row to value.

Overrides:

setValueAt in class

javax.swing.table.AbstractTableModel

Parameters:

value - the new value.

row - the row whose value is to be changed.

col - the column whose value is to be

changed.

project.logserver

Class ProcFrame

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Window
                    |
                    +--java.awt.Frame
                          |
                          +--javax.swing.JFrame
                                |
                                +--
```

project.logserver.ProcFrame

public class ProcFrame

extends javax.swing.JFrame

This class creates a new frame for displaying individual client's logging messages.

project.logserver

Class Properties

java.lang.Object

|

+--**project.logserver.Properties**

public class **Properties**

extends java.lang.Object

This class parses the information that is in the property file.

Constructor Detail

Properties

public **Properties**()

Method Detail

parseProperties

public void **parseProperties**(java.lang.String filename)

Parses the information that is in the property file.

Parameters:

filename - the String representing the name of the input file to read from.

project.logserver

Class TableMap

java.lang.Object

  |

  +--javax.swing.table.AbstractTableModel

        |

        +--**project.logserver.TableMap**

Direct Known Subclasses:

    TableSorter

public class TableMap

extends javax.swing.table.AbstractTableModel

implements javax.swing.event.TableModelListener

This class implements TableModel by routing all requests to its model, and TableModelListener by routing all events to its listeners. The source code is from the "The Java Tutorial" by Philip Milne.

Constructor Detail

TableMap

public **TableMap**()

    The TableMap default constructor.

Method Detail

getModel

public javax.swing.table.TableModel getModel()

Gets the TableModel.

Returns:

a TableModel.

setModel

public void **setModel**(javax.swing.table.TableModel model)

Sets the TableModel.

Parameters:

model - the TableModel.

getValueAt

public java.lang.Object **getValueAt**(int row,
int col)

Gets the value of a specific table cell.

Overrides:

getValueAt in class
javax.swing.table.AbstractTableModel

Parameters:

row - the row whose value is to be queried.

col - the column whose value is to be
queried.

Returns:

an Object at the specified cell.

setValueAt

public void **setValueAt**(java.lang.Object value,

int row,

int column)

Sets the value in the cell at col and row to value.

Overrides:

setValueAt in class

javax.swing.table.AbstractTableModel

Parameters:

value - the new value.

row - the row whose value is to be changed.

col - the column whose value is to be

changed.

getRowCount

public int **getRowCount**()

Gets the number of rows in the model.

Overrides:

getRowCount in class

javax.swing.table.AbstractTableModel

Returns:

an integer represents the number of rows in the model.

getColumnCount

public int **getColumnCount**()

Gets the number of columns in the model.

Overrides:

getColumnCount in class javax.swing.table.AbstractTableModel

Returns:

an integer represents the number of columns in the model.

getColumnName

public java.lang.String **getColumnName**(int aColumn)

Gets the name of the column at aColumn.

Overrides:

getColumnName in class javax.swing.table.**AbstractTableModel**

Parameters:

aColumn - the index of the column.

Returns:

a String representing the column name.

getColumnClass

public java.lang.Class **getColumnClass**(int aColumn)

Gets the most specific superclass for all the cell values in the column.

Overrides:

getColumnClass in class javax.swing.table.AbstractTableModel

Parameters:

aColumn - the index of the column.

Returns:

a class.

isCellEditable

public boolean **isCellEditable**(int row,
int column)

Returns true if the table editable, otherwise false.

Overrides:

isCellEditable in class javax.swing.table.AbstractTableModel

Parameters:

row - the row whose value to be queried.

column - the column whose value to be

queried.

Returns:

true if the cell is editable.

tableChanged

public void **tableChanged**(javax.swing.event.TableModelEvent e)

Forwards all events to all the listeners by default.

Specified by:

tableChanged in interface javax.swing.event.TableModelListener

Parameters:

e - TableEvent.

project.logserver

Class TableSorter

java.lang.Object

|

+--javax.swing.table.AbstractTableModel

|

+--project.logserver.TableMap

|

+--**project.logserver.TableSorter**

public class TableSorter

extends TableMap

A sorter for TableModels. The source code is from the "The Java Tutorial" by Philip Milne.

Constructor Detail

TableSorter

public **TableSorter**()

TableSort default constructor.

TableSorter

public **TableSorter**(javax.swing.table.TableModel model)

TableSorter constructor.

Parameters:

model - the Tablemodel.

Method Detail

setModel

public void **setModel**(javax.swing.table.TableModel model)

Sets the tablemoel.

Overrides:

setModel in class TableMap

Parameters:

model - the TableModel.

compareRowsByColumn

public int **compareRowsByColumn**(int row1,
int row2,
int column)

Compares the table rows by the current column. The returned integer of -1, means that the value in row1 is less than that in row2; 1 means that the value in row1 is greater than that in row2; and 0 means that the value in row1 is equal to that in row2.

Parameters:

row1 - the first row to be compare with.

row2 - the second row to be compare with.

column - the indext of the current column.

Returns:

an integer.

compare

public int **compare**(int row1,

int row2)

Compares values in two rows in the table. Returned value of 0 represents ascending order.

Parameters:

row1 - the first row to be compare with.

row2 - the second row to be compare with.

Returns:

0.

reallocateIndexes

public void **reallocateIndexes**()

Sets up a new array of indexes for the number of elements for the new data model.

tableChanged

public void **tableChanged**(javax.swing.event.TableModelEvent e)

Notifies that the table model changed.

Overrides:

tableChanged in class TableMap

Parameters:

e - the TableModelEvent.

checkModel

public void **checkModel**()

Checks if the table model has been changed.

sort

public void **sort**(java.lang.Object sender)

Sorts the table.

Parameters:

sender - the Object.

shuttlesort

public void **shuttlesort**(int[] from,
int[] to,
int low,
int high)

Sorts the data in the table.

Parameters:

from - the first half of array.

to - the end last half of array.

low - the beginning index.

high - the end index.

swap

public void **swap**(int i,

int j)

Swaps two values.

Parameters:

i - the index of array.

j - the index of array.

getValueAt

public java.lang.Object **getValueAt**(int row,

int col)

Gets the value for the cell at col and row.

Overrides:

getValueAt in class TableMap

Parameters:

row - the row whose value is to be queried.

col - the column whose value is to be queried.

Returns:

a value Object at the specified cell.

setValueAt

public void **setValueAt**(java.lang.Object value,

int row,

int col)

Sets the value in the cell at col and row to value.

Overrides:

setValueAt in class TableMap

Parameters:

value - the new value.

row - the row whose value is to be changed.

col - the column whose value is to be changed.

sortByColumn

public void **sortByColumn**(int column)

Sort the table by column.

Parameters:

column - the table column index.

sortByColumn

public void **sortByColumn**(int column,

boolean ascending)

Sorts the table by column.

Parameters:

column - the table column index.

ascending - true if by the asccending order.

addMouseListenerToHeaderInTable

public void

addMouseListenerToHeaderInTable(javax.swing.JTable table)

Add a mouse listener to the Table to trigger a table sort when a column heading is clicked in the JTable.

Parameters:

table - the JTable.

REFERENCES

[1] Deitel $ Deitel, "Java How to Program", Prentice Hall, Third Edition, 1999.

[2] Cay S. Hostmann & Gary Cornell "Core Java", The Sun Microsystems Press, 2000.

[3] David Flanagan, "Java Example in a Nutshell", O'Reilly, 2000

[4] "XML Programming", Hands On Technology Transfer, Inc., 2000.

[5] Log4j Web site for Log4j technology http://jakarta.apache.org/log4j

[6] The Source for Java Technology Web Site http://java.sun.com/docs/books/tutorial/uiswing

[7] The chainsaw web site http://www.puppycrawl.com/chainsaw.

[8] David Hunter, "Beginning XML", Wrox Press Ltd, 2000